SYBASE®

API Reference Manual

EAServer

Version 5.2

DOCUMENT ID: DC38037-01-0520-01

LAST REVISED: January 2005

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

This book, the *EAServer API Reference Manual*, contains reference pages for EAServer proprietary Java classes, C++ classes, ActiveX interfaces, and C routines. EAServer also supports many standard Java 2 Enterprise Edition (J2EE) and CORBA APIs. For information on these, see:

- The *EAServer Programmer's Guide* for usage information.
- The relevant standards document for API reference information. For J2EE standards documents, please see the Sun Microsystems J2EE Web pages at http://java.sun.com/. For CORBA standards documentation, please see the Object Management Group (OMG) Web site at http://www.omg.org/.

Audience

This book is written as a reference for developers of EAServer applications. Developers should know their development language and programming tools.

How to use this book

Chapter 1, "Java Classes and Interfaces" documents EAServer's Java classes and interfaces. You will need this information to implement Java components or Java clients.

Chapter 2, "ActiveX C++ Interface Reference" documents EAServer's ActiveX C++ interfaces. You will need this information to implement ActiveX components using C++.

Chapter 3, "ActiveX IDispatch Interface Reference" documents EAServer's ActiveX automation interfaces. You will need this information to implement ActiveX components using IDEs that use ActiveX automation such as Microsoft Visual Basic.

Chapter 4, "ActiveX Client Interfaces" documents the interfaces that ActiveX clients use to process result sets returned by a component method invocation.

Chapter 5, "C Routines Reference" documents EAServer's C library routines. You will need this information to implement C components.

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
commands and methods	When used in descriptive text, this font indicates keywords such as: • Command names used in descriptive text • C++ and Java method or class names used in descriptive text • Java package names used in descriptive text • Property names in the raw format, as when using jagtool to configure applications rather than EAServer Manager
variable, *package*, or *component*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: *Server*.log • File names • Names of components, EAServer packages, and other entities that are registered in the EAServer naming service
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates "select Save from the File menu."
`package 1`	Monospace font indicates: • Information that you enter in EAServer Manager, a command line, or as program text • Example program fragments • Example output fragments

Related documents

Core EAServer documentation The core EAServer documents are available in HTML format in your EAServer software installation, and in PDF and DynaText format on the *Technical Library* CD.

What's New in EAServer summarizes new functionality in this version.

The *EAServer Cookbook* contains tutorials and explains how to use the sample applications included with your EAServer software.

The *EAServer Feature Guide* explains application server concepts and architecture, such as supported component models, network protocols, server-managed transactions, and Web applications.

The *EAServer System Administration Guide* explains how to:

- Start the preconfigured Jaguar server and manage it with the EAServer Manager plug-in for Sybase Central™
- Create, configure, and start new application servers
- Define connection caches

- Create clusters of application servers to host load-balanced and highly available components and Web applications
- Monitor servers and application components
- Automate administration and monitoring tasks with jagtool

The *EAServer Programmer's Guide* explains how to:

- Create, deploy, and configure components and component-based applications
- Create, deploy, and configure Web applications, Java servlets, and JavaServer Pages
- Use the industry-standard CORBA and Java APIs supported by EAServer

The *EAServer Web Services Toolkit User's Guide* describes Web services support in EAServer, including:

- Support for standard Web services protocols such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Uniform Description, Discovery, and Integration (UDDI)
- Administration tools for deployment and creation of new Web services, WSDL document creation, UDDI registration, and SOAP management

The *EAServer Security Administration and Programming Guide* explains how to:

- Understand the EAServer security architecture
- Configure role-based security for components and Web applications
- Configure SSL certificate-based security for client connections using the Security Manager plug-in for Sybase Central
- Implement custom security services for authentication, authorization, and role membership evaluation
- Implement secure HTTP and IIOP client applications
- Deploy client applications that connect through Internet proxies and firewalls

The *EAServer Performance and Tuning Guide* describes how to tune your server and application settings for best performance.

The *EAServer Troubleshooting Guide* describes procedures for troubleshooting problems that EAServer users may encounter. This document is available only online; see the EAServer Troubleshooting Guide at http://www.sybase.com/detail?id=1024509.

Message Bridge for Java™ Message Bridge for Java simplifies the parsing and formatting of structured documents in Java applications. Message Bridge allows you to define structures in XML or other formats, and generates Java classes to parse and build documents and messages that follow the format. The *Message Bridge for Java User's Guide* describes how to use the Message Bridge tools and runtime APIs. This document is included in PDF and DynaText format on your *EAServer Technical Library* CD.

Adaptive Server Anywhere documents EAServer includes a limited-license version of Adaptive Server Anywhere for use in running the samples and tutorials included with EAServer. Adaptive Server Anywhere documents are available on the Sybase Web site at http://sybooks.sybase.com/aw.html.

jConnect for JDBC documents EAServer includes the jConnect™ for JDBC™ driver to allow JDBC access to Sybase database servers and gateways. The *Programmer's Reference jConnect for JDBC* is available on the Sybase Web site at http://sybooks.sybase.com/jc.html.

Accessibility features

EAServer has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in HTML, JavaHelp, and Eclipse help formats, which you can navigate using a screen reader.

EAServer Manager supports working without a mouse. For more information, see "Keyboard navigation" in Chapter 2, "Sybase Central Overview," in the *EAServer System Administration Guide*.

The WST plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired or have other special needs. For information about these features refer to Eclipse help:

1 Start Eclipse

2 Select Help | Help Contents

3 Enter Accessibility in the Search dialog box

4 Select Accessible user interfaces or Accessibility features for Eclipse

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Select Products from the navigation bar on the left.

3 Select a product name from the product list and click Go.

4 Select the Certification Report filter, specify a time frame, and click Go.

5 Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Java Classes and Interfaces

Package index

com.sybase.CORBA.jdbc102

For use in classes that will be run in a JDK-1.0.2-compatible Java virtual machine. Classes in this package use the same names and method syntax as com.sybase.CORBA.jdbc11, except for the following package substitutions:

JDK 1.0.2 package	Substitutes for
jdbc.math	java.math
jdbc.sql	java.sql

Most programmers use import statements to determine whether JDK 1.0.2 or JDK 1.1 versions of the classes are used.

com.sybase.CORBA.jdbc11

For use in classes that will be run in a JDK-1.1-compatible Java virtual machine. Provides classes for converting between EAServer's predefined IDL datatypes and the core Java language objects:

- IDL – Provides methods to convert core Java datatypes to EAServer's predefined CORBA IDL datatypes.
- IdlResultSet – Implements the JServerResultSet interface, allowing you to construct TabularResults.ResultSet instances for component methods that return row results.
- SQL – Provides methods to convert EAServer's predefined CORBA IDL datatypes to core Java datatypes.

com.sybase.jaguar.jcm

Classes and interfaces for managing cached JDBC connections in server-side Java code:

- jaguar.jcm.JCM class – Provides access to JDBC connection caches that have been defined in EAServer Manager.
- jaguar.jcm.JCMCache class – Manages a pool of JDBC connections to a third-tier database server.
- jaguar.jcm.JConnectionNotFoundException class – Exception thrown when no connections are available.

com.sybase.jaguar.server

Utility classes used in server-side Java code:

- jaguar.server.Jaguar class – Provides utility methods for use in server-side Java code.
- jaguar.server.JContext class – Instantiates objects that are used to send result sets from a Java component method and provides a method to retrieve rows from a java.sql.ResultSet and forward them to the client.

com.sybase.jaguar.sql

Interfaces for objects that construct and send row results from a Java server component to the client:

- jaguar.sql.JServerResultSet interface – Provides methods to return result rows to a client application. JServerResultSet is similar to the java.sql.ResultSet interface, which is used to retrieve result rows from a server.
- jaguar.sql.JServerResultSetMetaData interface – Provides methods for describing the metadata of a result set. Metadata specifies the number of columns in each row as well as the datatype, format, nullability, and so forth for each column.

com.sybase.jaguar.util

Utility classes that are used in both server-side and client side Java code:

- jaguar.util.JException class – JException is the generic exception that is thrown by methods in the EAServer classes or in generated client stub classes.
- jaguar.util.<object>Holder class – Holder classes are used to pass INOUT parameters to component method calls. Each holder class has a *value* field that contains instances of a specific object or base Java type.

com.sybase.jaguar.util.jdbc102

Holder classes for use in code that will run in a version 1.0.2 Java virtual machine:

- jaguar.util.jdbc102.<object>Holder class – The com.sybase.jaguar.util.jdbc11 holder classes are used to pass jdbc.sql and jdbc.math objects as INOUT parameters. Use these classes in code that runs in a JDK 1.1 or later virtual machine.

  The classes in com.sybase.jaguar.util.jdbc102 and com.sybase.jaguar.util.jdbc11 have identical names and method signatures. You can switch between these classes simply by changing the import statements in your source files.

com.sybase.jaguar.util.jdbc11

Holder classes for use in code that will run in a version 1.1 or later Java virtual machine:

- jaguar.util.jdbc11.<object>Holder class – The com.sybase.jaguar.util.jdbc11 holder classes are used to pass java.sql and java.math objects as INOUT parameters. Use these classes in code that runs in a JDK 1.1 or later virtual machine.

com.sybase.CORBA.jdbc11.IDL class

Description package com.sybase.CORBA.jdbc11;

public abstract class IDL

Provides methods to convert core Java datatypes to EAServer's predefined CORBA IDL datatypes.

Constructors

None. All methods are static.

Methods

- getDate(java.sql.Date) – Converts a java.sql.Date object to an equivalent MJD::Date CORBA IDL object.
- getDecimal(java.math.BigDecimal) – Converts a BigDecimal object to an equivalent BCD::Decimal CORBA IDL object.
- getMoney(java.math.BigDecimal) – Converts a BigDecimal object to an equivalent BCD::Money CORBA IDL object.
- getResultSet(java.sql.ResultSet) – Converts a java.sql.ResultSet object to an equivalent TabularResults::ResultSet CORBA IDL object.
- getTime(java.sql.Time) – Converts a java.sql.Time object to an equivalent MJD::Time CORBA IDL object.
- getTimestamp(java.sql.Timestamp) – Converts a java.sql.Timestamp object to an equivalent MJD::Timestamp CORBA IDL object.

See also

com.sybase.CORBA.jdbc11.SQL class

IDL.getDate(java.sql.Date)

Description

Converts a java.sql.Date object to an equivalent MJD::Date CORBA IDL object.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	IDL

public static MJD.Date getDate(java.sql.Date value)

Parameters

value

A java.sql.Date value to be converted.

Return value

The value converted to an equivalent CORBA IDL MJD::Date value.

See also

getTime(java.sql.Time), getTimestamp(java.sql.Timestamp), SQL.getDate(MJD.Date)

IDL.getDecimal(java.math.BigDecimal)

Description

Converts a BigDecimal object to an equivalent BCD::Decimal CORBA IDL object.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	IDL

```
public static BCD.Decimal
    getDecimal(java.math.BigDecimal value)
    throws org.omg.CORBA.DATA_CONVERSION
```

Parameters

value

A java.math.BigDecimal value to be converted.

Return value

The value converted to an equivalent CORBA IDL BCD::Decimal value.

See also

getMoney(java.math.BigDecimal), SQL.getBigDecimal(BCD.Decimal)

IDL.getMoney(java.math.BigDecimal)

Description

Converts a BigDecimal object to an equivalent BCD::Money CORBA IDL object.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	IDL

```
public static BCD.Money getMoney(
     java.math.BigDecimal value)
    throws org.omg.CORBA.DATA_CONVERSION
```

Parameters

value

A java.math.BigDecimal value to be converted.

Return value

The value converted to an equivalent CORBA IDL BCD::Money value.

See also

getDecimal(java.math.BigDecimal), SQL.getBigDecimal(BCD.Money)

IDL.getResultSet(java.sql.ResultSet)

Description

Converts a java.sql.ResultSet object to an equivalent TabularResults::ResultSet CORBA IDL object.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	IDL

```
public static MJD.ResultSet
    getResultSet( java.sql.ResultSet rs)
```

Parameters

rs
A java.sql.ResultSet value to be converted.

Return value

The value converted to an equivalent CORBA IDL TabularResults::ResultSet value.

See also

SQL.getResultSet(TabularResults.ResultSet)

IDL.getTime(java.sql.Time)

Description

Converts a java.sql.Time object to an equivalent MJD::Time CORBA IDL object.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	IDL

```
public static MJD.Time getTime(java.sql.Time value)
```

Parameters

value
A java.sql.Time value to be converted.

Return value

The value converted to an equivalent CORBA IDL MJD::Time value.

See also

getDate(java.sql.Date), getTimestamp(java.sql.Timestamp), SQL.getTime(MJD.Time)

IDL.getTimestamp(java.sql.Timestamp)

Description

Converts a java.sql.Timestamp object to an equivalent MJD::Timestamp CORBA IDL object.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	IDL

```
public static MJD.Timestamp
    getTimestamp( java.sql.Timestamp value)
```

Parameters *value*
A java.sql.Timestamp value to be converted.

Return value The value converted to an equivalent CORBA IDL MJD::Timestamp value.

See also getDate(java.sql.Date), getTime(java.sql.Time), SQL.getTimestamp(MJD.Timestamp)

com.sybase.CORBA.jdbc11.IdlResultSet

Description

```
package com.sybase.CORBA.jdbc11;

public class IdlResultSet
  extends java.lang.Object
  implements jaguar.sql.JServerResultSet;
```

Implements the JServerResultSet interface, allowing you to construct TabularResults.ResultSet instances for component methods that return row results.

Component methods that return row results to clients return TabularResults.ResultSet or TabularResults.ResultSet[]. IdlResultSet allows you to create instances of these types using the JDBC style JServerResultSet interfaces.

For documentation of the TabularResults IDL types, see the generated Interface Repository documentation at ../../ir/TabularResults.html.

To return a single result set, initialize the rows and columns using the JServerResultSetMetaData and JServerResultSet methods, then convert to a TabularResults.ResultSet instance as shown in this code fragment:

```
JServerResultSetMetaData jsrs;
... define column formats ...
IdlResultSet irs = new IdlResultSet(jsrsmd);
... define row data using JServerResultSet methods ...
return irs.getResultSet();
```

To return multiple result sets, build an array of TabularResults.ResultSet instances, as follows:

1 Declare a java.util.Vector instance:

```
java.util.Vector vector = new Vector();
```

2 Initialize each IdlResultSet instance as described above, then add it to the vector:

```
    vector.addElement(irs.getResultSet());
```

3 When done, convert the vector to an array to be returned by the method:

```
TabularResults.ResultSet[] array =
    new TabularResults.ResultSet[vector.size()];
vector.copyInto(array);
return array;
```

Constructors

- IdlResultSet(java.sql.ResultSetMetaData) – Construct an instance using the column formats specified by a JServerResultSetMetaData instance. You can add rows to the instance using the JServerResultSet methods.
- IdlResultSet(java.sql.ResultSet) – Construct an instance by reading the rows from the supplied ResultSet.

Methods

- getResultSet() – Translate the contents of this instance into TabularResults.ResultSet instance.

See also

jaguar.sql.JServerResultSet interface, jaguar.sql.JServerResultSetMetaData interface

com.sybase.CORBA.jdbc11.SQL class

Description

package com.sybase.CORBA.jdbc11;

public abstract class SQL

Provides methods to convert EAServer's predefined CORBA IDL datatypes to core Java datatypes.

Constructors

None. All methods are static.

Methods

- getBigDecimal(BCD.Decimal) – Converts a BCD::Decimal CORBA IDL object to an equivalent java.math.BigDecimal.
- getBigDecimal(BCD.Money) – Converts a BCD::Money CORBA IDL object to an equivalent java.math.BigDecimal.
- getDate(MJD.Date) – Converts an MJD::Date CORBA IDL object to an equivalent java.sql.Date object.
- getResultSet(TabularResults.ResultSet) – Converts a TabularResults::ResultSet CORBA IDL object to an equivalent java.sql.ResultSet object.
- getTime(MJD.Time) – Converts an MJD::Time CORBA IDL object to an equivalent java.sql.Time object.

- getTimestamp(MJD.Timestamp) – Converts an MJD::Timestamp CORBA IDL object to an equivalent java.sql.Timestamp object.

See also com.sybase.CORBA.jdbc11.IDL class

SQL.getBigDecimal(BCD.Decimal)

Description Converts a BCD::Decimal CORBA IDL object to an equivalent java.math.BigDecimal.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	SQL

```
public static java.math.BigDecimal
    getBigDecimal(BCD.Decimal value)
```

Parameters *value*
A BCD.Decimal value to be converted.

Return value The value converted to an equivalent java.math.BigDecimal value.

See also getBigDecimal(BCD.Decimal), getBigDecimal(BCD.Money), IDL.getDecimal(java.math.BigDecimal)

SQL.getBigDecimal(BCD.Money)

Description Converts a BCD::Money CORBA IDL object to an equivalent java.math.BigDecimal.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	SQL

```
public static java.math.BigDecimal
    getBigDecimal(BCD.Money value)
```

Parameters *value*
A BCD.Money value to be converted.

Return value The value converted to an equivalent java.math.BigDecimal value.

See also getBigDecimal(BCD.Decimal), IDL.getMoney(java.math.BigDecimal)

SQL.getDate(MJD.Date)

Description

Converts an MJD::Date CORBA IDL object to an equivalent java.sql.Date object.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	SQL

```
public static java.sql.Date getDate(MJD.Date value)
```

Parameters

value

An MJD::Date value to be converted.

Return value

The value converted to an equivalent java.sql.Date value.

See also

getTime(MJD.Time), getTimestamp(MJD.Timestamp), IDL.getDate(java.sql.Date)

SQL.getResultSet(TabularResults.ResultSet)

Description

Converts a TabularResults::ResultSet CORBA IDL object to an equivalent java.sql.ResultSet object.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	SQL

```
public static java.sql.ResultSet
    getResultSet(TabularResults.ResultSet rs)
```

Parameters

rs

A TabularResults.ResultSet object to be converted.

Return value

The value converted to an equivalent java.sql.ResultSet value.

See also

IDL.getResultSet(java.sql.ResultSet)

SQL.getTime(MJD.Time)

Description

Converts an MJD::Time CORBA IDL object to an equivalent java.sql.Time object.

Syntax

Package	com.sybase.CORBA.jdbc11

Class	SQL

public static java.sql.Time getTime(MJD.Time value)

Parameters

value
An MJD.Time value to be converted.

Return value

The value converted to an equivalent java.sql.Time value.

See also

getDate(MJD.Date), getTimestamp(MJD.Timestamp), IDL.getTime(java.sql.Time)

SQL.getTimestamp(MJD.Timestamp)

Description

Converts an MJD::Timestamp CORBA IDL object to an equivalent java.sql.Timestamp object.

Syntax

Package	com.sybase.CORBA.jdbc11
Class	SQL

public static java.sql.Timestamp
getTimestamp(MJD.Timestamp value)

Parameters

value
An MJD.Timestamp value to be converted.

Return value

The value converted to an equivalent java.sql.Timestamp value.

See also

getDate(MJD.Date), getTime(MJD.Time), IDL.getTimestamp(java.sql.Timestamp)

jaguar.jcm.JCM class

Description

package com.sybase.jaguar.jcm;
public class JCM extends Object

Provides access to JDBC connection caches that have been defined in EAServer Manager.

Constructors

None. All methods are static.

Methods

- byNameAllowed(String) – Determines if a cache can be retrieved by calling getCacheByName(String).

- getCache(String, String, String) – Returns a reference to a connection cache with matching values for the specified user name, password, and server name.
- getCacheByName(String) – Returns a reference to the connection cache with the given name.

Usage

For an introduction to the Java connection management classes, see Chapter 26, "Using Connection Management," in the *EAServer Programmer's Guide*.

JCM.byNameAllowed(String)

Description

Determines if a cache can be retrieved by calling getCacheByName(String).

Syntax

Package	com.sybase.jaguar.jcm
Interface	JCM

```
public static boolean byNameAllowed
   (String name)
   throws JException
```

Parameters

name

The name of the cache of interest, as entered in EAServer Manager.

Return value

`true` if a cache is installed with the specified name, and the cache can be retrieved with JCM.getCacheByName(String); `false` otherwise.

Usage

The getCacheByName(String) method allows you to retrieve a connection cache by specifying only the cache name, rather than specifying values for the cache user name, password, and server name. However, by-name access must be enabled for the cache in EAServer Manager to allow retrieval with getCacheByName(String).

You can call byNameAllowed to determine whether by-name access is allowed for a specified cache.

See also

getCacheByName(String)

JCM.getCache(String, String, String)

Description

Returns a reference to a connection cache with matching values for the specified user name, password, and server name.

Syntax

Package	com.sybase.jaguar.jcm
Interface	JCM

```
public static JCMCache getCache
   ( String user, String pwd, String server)
   throws JException
```

Parameters

user

The database user name associated with the cache.

pwd

The database password associated with the cache.

server

The database server name associated with the cache. The value should be a JDBC connection URL in the appropriate format for calls to java.sql.DriverManager.getConnection(String). The URL format depends on which JDBC driver the cache uses. See your JDBC driver documentation for more information.

Return value

A reference to a JCMCache instance with matching values for *user*, *pwd,* and *server*.

A JException exception is thrown if no cache with matching values exists.

Usage

The supplied values for *user*, *pwd*, and *server* must match the properties of an existing cache.

See also

Chapter 4, “Database Access,” in the *EAServer System Administration Guide*

getCacheByName(String)

JCM.getCacheByName(String)

Description

Returns a reference to the connection cache with the specified name.

Syntax

Package	com.sybase.jaguar.jcm
Interface	JCM

```
public static JCMCache getCacheByName
   ( String name)
   throws JException
```

Parameters

name

The name of the cache to be retrieved, as entered in EAServer Manager.

Return value

A reference to a JCMCache instance with a matching value for *name*.

A JException exception is thrown if:

- No cache is installed with the specified name.
- A matching cache is installed, but the cache properties forbid retrieval with this method. Use getCache(String, String, String) instead.

Usage

getCacheByName allows you to retrieve a connection cache by specifying only the cache name, rather than specifying values for the cache user name, password, and server name.

Using this method rather than getCache(String, String, String) allows you to change the cache user name, password, or server in EAServer Manager without requiring corresponding changes to your component source code.

In order for components to retrieve a cache with getCacheByName, the EAServer Administrator must select the "Enable cache-by-name access" option for the cache in EAServer Manager. getCacheByName throws an exception if the cache does not have this option enabled.

See also

Chapter 4, "Database Access," in the *EAServer System Administration Guide*

getCacheByName(String), getCache(String, String, String)

jaguar.jcm.JCMCache class

Description

```
package com.sybase.jaguar.jcm;
public class JCMCache extends Object
```

Manages a pool of connections to a third-tier database server.

Constructors

None. Call JCM.getCache(String, String, String).

Fields

JCM_FORCE

```
public final static int JCM_FORCE
```

A value for the getConnection *flag* parameter.

JCM_NOWAIT

```
public final static int JCM_NOWAIT
```

A value for the getConnection *flag* parameter.

JCM_WAIT

public final static int JCM_WAIT

A value for the getConnection *flag* parameter.

Methods

- byNameAllowed() – Determines whether the cache can be retrieved by calling JCM.getCacheByName(String).
- dropConnection(Connection) – Drops a connection. The connection is closed and not released into the cache.
- getPoolSizeMax() – Retrieves the maximum number of connections that this cache can manage.
- getConlibName() – Returns the connectivity library (or interface) name for the cache.
- getConnection(int) – Obtains a connection handle from the cache.
- getProxyConnection(int, String) – Obtains a connection handle from the cache, specifying an alternate login name to set-proxy to.
- getName() – Retrieves the cache's name.
- getPassword() – Retrieves the password used by connections in the cache.
- getRemoteServerName() – Returns the remote server name used by connections in the cache.
- getUsername() – Retrieves the user name used by connections in the cache.
- releaseConnection(Connection) – Releases a connection to the cache for reuse.

See also

java.sql.Connection, "Using Java Connection Manager classes" in the *EAServer Programmer's Guide*

JCMCache.byNameAllowed()

Description

Determines whether the cache can be retrieved by calling JCM.getCacheByName(String).

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

public boolean byNameAllowed()

Return value — `true` if the cache can be retrieved with JCM.getCacheByName(String), `false` otherwise.

Usage — The "Enable cache-by-name access" option in the Connection Cache Properties dialog determines whether components can retrieve the cache by calling JCM.getCacheByName(String). See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.

See also — getName(), JCM.byNameAllowed(String), JCM.getCacheByName(String)

JCMCache.dropConnection(Connection)

Description — Drops a connection. The connection is closed and not released into the cache.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public void dropConnection( Connection con)
   throws SQLException
```

Parameters — *con*
The java.sql.Connection instance to be dropped.

Usage — Use dropConnection() to close a connection when you do not want the connection returned to the cache. If necessary, future getConnection(int) calls will allocate new connections to replace any that have been dropped.

See also — getConnection(int), releaseConnection(Connection)

JCMCache.getConlibName()

Description — Returns the connectivity library (or interface) name for the cache.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public String getConlibName()
```

Return value — "JDBC"

JCMCache.getConnection(int)

Description

Obtains a connection handle from the cache.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public Connection getConnection(int flag)
  throws SQLException, JException,
  JConnectionNotFoundException
```

Parameters

flag

A symbolic value that specifies what should happen if the maximum number of connections have been allocated and are in use (that is, no connection is available in the cache). Allowable values are:

Value	Behavior when no connection is available
JCM_NOWAIT	Throws JConnectionNotFoundException.
JCM_WAIT	Does not return until a cached connection is available.
JCM_FORCE	"Forces" open a new, uncached connection. The cache's maximum size is ignored.

Return value

A java.sql.Connection instance from the connection cache. If the call specifies JCM_NOWAIT and no connections are available, the call throws a JConnectionNotFoundException instance.

Usage

getConnection(int) attempts to return a connection from the cache. Caches are maintained statically; a cache is initially empty when the server starts. Subsequent getConnection(int) calls allocate connections when necessary. releaseConnection(Connection) calls release control of a connection for later reuse.

Each cache has a maximum number of connections determined by the cache's definition in EAServer Manager. (See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.) The *flag* parameter determines getConnection(int) behavior when the cache's maximum number of connections are in use. getPoolSizeMax() returns the cache's maximum number of connections.

For improved performance, connections should not be held any longer than necessary. As a general rule, methods that use a cached connection should release it with releaseConnection(Connection) before returning. This strategy minimizes contention by multiple components for a cache's connections.

See also

dropConnection(Connection), getPoolSizeMax(), releaseConnection(Connection)

JCMCache.getPoolSizeMax()

Description — Retrieves the maximum number of connections that can be pooled in the cache.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public int getPoolSizeMax()
```

Return value — The cache size.

Usage — The size of a cache is specified the Connection Cache Properties in EAServer Manager. See Chapter 4, “Database Access,” in the *EAServer System Administration Guide* for more information.

See also — getPoolSizeMin()

JCMCache.getPoolSizeMin()

Description — Retrieves the maximum number of connections that can be pooled in the cache.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public int getPoolSizeMax()
```

Return value — The cache size.

Usage — The size of a cache is specified the Connection Cache Properties in EAServer Manager. See Chapter 4, “Database Access,” in the *EAServer System Administration Guide* for more information.

See also — getPoolSizeMax()

JCMCache.getProxyConnection(int, String)

Description — Obtains a connection handle from the cache, specifying an alternate login name to set-proxy to.

Not all connection caches support set-proxy
Set-proxy support must be enabled for caches in EAServer Manager before you can use this feature. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information. You must be connected to a database server, such as Adaptive Server Enterprise 11.5 or later, that supports the set session authorization command.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public Connection getProxyConnection(int flag, String proxy)
   throws SQLException, JException,
  JConnectionNotFoundException
```

Parameters

flag

A symbolic value that specifies what should happen if the maximum number of connections have been allocated and are in use (that is, no connection is available in the cache). Allowable values are:

Value	Behavior when no connection is available
JCM_NOWAIT	Throws JConnectionNotFoundException.
JCM_WAIT	Does not return until a cached connection is available.
JCM_FORCE	"Forces" open a new, uncached connection. The cache's maximum size is ignored.

proxy

The user name to set-proxy to.

Return value

A java.sql.Connection instance from the connection cache. If the call specifies JCM_NOWAIT and no connections are available, the call throws a JConnectionNotFoundException instance.

Usage

This method retrieves a cached connection, specifying an alternate login name to set-proxy to. Set-proxy support must be enabled for a cache in EAServer Manager. If support is enabled, connections retrieved from the cache with getConnection(int) set-proxy to the client user name. Call getProxyConnection(int, String) to specify a different user name to set-proxy to.

Other than the set-proxy behavior, getProxyConnection(int, String) is identical to getConnection(int).

See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for information on defining caches and enabling set-proxy support.

For improved performance, connections should not be held any longer than necessary. As a general rule, methods that use a cached connection should release it with releaseConnection(Connection) before returning. This strategy minimizes contention by multiple components for a cache's connections.

See also: dropConnection(Connection), getPoolSizeMax(), getConnection(int), releaseConnection(Connection)

JCMCache.getName()

Description: Retrieves the cache's name.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public String getName()
```

Return value: The cache's name.

Usage: You can change a cache's name using EAServer Manager. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.

JCMCache.getPassword()

Description: Retrieves the password used by connections in the cache.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public String getPassword()
```

Return value: The password.

Usage: A cache's password is specified on the cache's EAServer Manager property sheet. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.

See also: getRemoteServerName(), getUsername()

JCMCache.getRemoteServerName()

Description: Retrieves the remote server name used by connections in the cache.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public String getRemoteServerName()
```

Return value: The remote server name.

Usage: A cache's remote server name is specified on the cache's EAServer Manager property sheet. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.

See also: getPassword(), getUsername()

JCMCache.getUserName()

Description: Retrieves the user name used by connections in the cache.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public String getUserName()
```

Return value: The user name.

Usage: A cache's user name is specified on the cache's EAServer Manager property sheet. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.

See also: getPassword(), getRemoteServerName()

JCMCache.releaseConnection(Connection)

Description: Releases a connection to the cache for reuse.

Syntax

Package	com.sybase.jaguar.jcm
Class	JCMCache

```
public void releaseConnection( Connection con)
  throws SQLException
```

Parameters

con
The connection to release.

Usage

Released connections must be in a state that allows new queries to be issued.

The connection will be dropped (and not returned to the cache) if the cache has exceeded its maximum number of connections. The maximum number of connections can be exceeded if calls to getConnection(int) are issued with *flag* as JCM_FORCE. In this case, releaseConnection drops the excess connections.

Many JDBC programs do not explicitly clean up java.sql.Statement objects. Instead, they rely on the JDBC driver to clean up Statement objects when the connection is closed. This strategy does not work with cached connections: you must explicitly clean up Statement objects before releasing a connection back into the cache. To clean up Statement objects, call Statement.close() and set the Statement reference to `null`.

Warning! To prevent memory leaks, you must explicitly clean up a connection's Statement objects before releasing the connection back into the cache. Do not release a connection more than once.

See also

getConnection(int), dropConnection(Connection)

jaguar.jcm.JConnectionNotFoundException class

Description

```
package com.sybase.jaguar.jcm;
public class JConnectionNotFoundException
  extends JException;
```

Exception thrown by JCMCache.getConnection(int) to indicate that no connections are available in the cache. You must specify JCM_NOWAIT in order for the exception to be thrown.

Constructors

Same as JException.

Methods

Same as JException.

See also

jaguar.util.JException class, java.sql.SQLException class

jaguar.server.Jaguar class

Description

```
package com.sybase.jaguar.server;
public class Jaguar extends Object
```

Provides utility methods for use in server-side Java code.

Constructors

None. All methods are static.

Methods

- getInstanceContext() – Returns the InstanceContext object associated with the current component instance.
- getHostName() – Returns the client host name for the client connection that is associated with this component instance.
- getPassword() – Returns the password for the client connection that is associated with this component instance.
- getPeerAddress() – Returns the client host address for the client connection that is associated with this component instance.
- getServerName() – Returns the name of the server.
- getUserName() – Returns the user name for the client connection that is associated with this component instance.
- inJaguar() – Tests if running inside the server.
- writeLog(boolean, String) – Writes a message to the server's log file.

Jaguar.getInstanceContext()

Description

Retrieves the InstanceContext object associated with the current component instance.

Syntax

Package	com.sybase.jaguar.server
Class	Jaguar

```
public InstanceContext getInstanceContext()
```

Return value

An InstanceContext object for the current component instance.

Usage

Components that do not implement the ServerBean interface can call this method to get an InstanceContext object. The InstanceContext provides transaction primitives that allow the component to influence the outcome of the transactions in which it participates.

Components that implement InstanceContext receive the InstanceContext via the ServerBean.activate(InstanceContext, String) method.

See also InstanceContext, ServerBean

Jaguar.getHostName()

Description Returns the client host name for the client connection that is associated with this component instance.

Syntax

Package	com.sybase.jaguar.server
Class	Jaguar

public static String getHostName() throws JException

Return value The client host name. The host name can be 0 length if the client software did not supply the host name.

Note
Java clients do not supply the client host name (there is no mechanism to retrieve the host name in Java).

See also getPeerAddress()

Jaguar.getPassword()

Description Returns the password for the client connection that is associated with this component instance.

Syntax

Package	com.sybase.jaguar.server
Class	Jaguar

public static String getPassword() throws JException

Return value The client password. The password can be 0 length.

Usage getPassword returns the password for the client connection that is associated with this component instance.

This method cannot be called from a component instance that is running as a service component, since service components run without client interaction.

See also getUserName()

Jaguar.getPeerAddress()

Description Returns the client host address for the client connection that is associated with this component instance.

Syntax

Package	com.sybase.jaguar.server
Class	Jaguar

public static String getPeerAddress() throws JException

Return value The client's IP address, or "0.0.0.0" if the client's IP address is unavailable.

See also getHostName()

Jaguar.getServerName()

Description Returns the name of the server.

Syntax

Package	com.sybase.jaguar.server
Class	Jaguar

public static String getServerName() throws JException

Return value The name of the server.

Jaguar.getUserName()

Description Returns the user name for the client connection that is associated with this component instance.

Syntax

Package	com.sybase.jaguar.server
Class	Jaguar

public static String getUserName() throws JException

Return value The user name. The user name can be 0 length.

Usage

getUserName returns the user name for the client connection that is associated with this component instance.

This method cannot be called from a component instance that is running as a service component, since service components run without client interaction.

See also

getPassword()

Jaguar.inJaguar()

Description

Tests if running inside the server.

Syntax

Package	com.sybase.jaguar.server
Class	Jaguar

```
public static boolean inJaguar() throws JException
```

Return value

true if running inside the server, false otherwise.

Usage

As an alternative, you can call the method com.sybase.CORBA.ORB.isClient(), which returns a boolean value that is true if running outside of EAServer. Use this alternative if your code may be run without the EAServer server-side classes in the CLASSPATH.

Jaguar.writeLog(boolean, String)

Description

Writes a message to the server's log file.

Standard output redirected to the server log
Prehistoric EAServer versions required you to call this method to write to the log. In version 3.0 or later, you can call any of the System.out.print methods.

Syntax

Package	com.sybase.jaguar.server
Class	Jaguar

```
public static native void writeLog
   (boolean use_date, String logmsg)
   throws JException
```

Parameters

use_date

`true` if the current date and time should be prepended to the log message; `false` otherwise.

logmsg

A message to be written to the server's log file.

Usage

This method records a message in the server's log file.

By convention, errors that occur on the server are written to the log. Java components should call writeLog(String) rather than printing to the console with java.lang.System.out or java.lang.System.err.

For information on configuring the log file used by the server, see Chapter 3, "Creating and Configuring Servers," in the *EAServer System Administration Guide*.

jaguar.server.JContext class

Description

package com.sybase.jaguar.server;

public class JContext extends Object

Instantiates objects that are used to send result sets from a Java component method and provides a method to forward rows from a java.sql.ResultSet to the client.

Constructors

None. All methods are static.

Methods

- createServerResultSetMetaData() – Creates a JServerResultSetMetaData object.
- createServerResultSet(JServerResultSetMetaData) – Creates a JServerResultSet object with row format that matches the specified JServerResultSetMetaData object.
- forwardResultSet(ResultSet) – Retrieves the rows from a java.sql.ResultSet object and forward them to the client.
- getComponentName() – Retrieves the name of the currently executing component, as displayed in EAServer Manager.
- getPackageName() – Determines the name of the package in which the currently executing component is installed.

See also

JServerResultSet, JServerResultSetMetaData

JContext.createServerResultSetMetaData()

Description

Creates a JServerResultSetMetaData object.

Syntax

Package	com.sybase.jaguar.server
Class	JContext

```
public static JServerResultSetMetaData
    createServerResultSetMetaData()
  throws SQLException
```

Usage

The JServerResultSetMetaData reference can be used to describe result rows to be sent to the client.

See also

createServerResultSet(JServerResultSetMetaData), forwardResultSet(ResultSet)

JContext.createServerResultSet(JServerResultSetMetaData)

Description

Creates a JServerResultSet object.

Syntax

Package	com.sybase.jaguar.server
Class	JContext

```
public static JServerResultSet  createServerResultSet
  ( JServerResultSetMetaData metadata)
  throws SQLException
```

Parameters

metadata

A JServerResultSetMetaData object that has been initialized to describe the result set that will be sent.

See also

createServerResultSetMetaData(), forwardResultSet(ResultSet)

JContext.forwardResultSet(ResultSet)

Description

Retrieves the rows from a java.sql.ResultSet object and forward them to the client.

Syntax

Package	com.sybase.jaguar.server
Class	JContext

```
public static void
    forwardResultSet( ResultSet rs)
   throws SQLException
```

Parameters

rs

A java.sql.ResultSet containing result rows from a JDBC query to a third-tier server.

See also

java.sql.ResultSet

JContext.getComponentName()

Description

Retrieves the name of the currently executing component, as displayed in EAServer Manager.

Syntax

Package	com.sybase.jaguar.server
Class	JContext

```
public static String
    getComponentName()
```

Return value

The name of the component, as displayed in EAServer Manager.

Usage

getPackageName() and getComponentName() allow you to determine the name of the currently executing component. Within a server, components are identified by the name of the EAServer Manager package where they are installed and the EAServer Manager component name.

See also

getPackageName(), Jaguar.getServerName()

JContext.getPackageName()

Description

Determines the name of the package in which the currently executing component is installed.

Syntax

Package	com.sybase.jaguar.server
Class	JContext

```
public static String
    getPackageName()
```

Return value

The name of the EAServer package, as displayed in EAServer Manager.

Usage

getPackageName() and getComponentName() allow you to determine the name of the currently executing component. Within a server, components are uniquely identified by the name of the EAServer Manager package where they are installed and the EAServer Manager component name.

See also

getComponentName(), Jaguar.getServerName()

jaguar.sql.JServerResultSet interface

Description

package com.sybase.jaguar.sql;

public interface JServerResultSet extends Object

Provides methods to send rows to the client. JServerResultSet is similar to the java.sql.ResultSet interface, which is used to retrieve result rows from a server.

Constructors

Call JContext.createServerResultSet(JServerResultSetMetaData).

Methods

- done() – Indicates that all rows in a result set have been sent.
- findColumn(String) – Maps a column name to a column index.
- getMetaData() – Returns a java.sql.ResultSetMetaData object that describes the rows in a result set. The metadata includes the number of columns, the datatype of each column, and other details about each column such as whether values can be NULL.
- next() – Sends a row to the client.
- setBigDecimal(int, BigDecimal, int) – Specifies a non-NULL value for a BigDecimal column.
- setCurrency(int, long) – Specifies a non-NULL value for a column that represents a cash value.
- setNull(int) – Specifies that a column in the current row has value NULL.
- set<Object>(int, <Object>) – Specifies a non-NULL value for a column in the current row.

Usage

A JServerResultSetMetaData instance is required to construct a JServerResultSet. JServerResultSetMetaData describes the format of rows in the result set. After initializing the JServerResultSetMetaData instance, call JContext.createServerResultSet(JServerResultSetMetaData).

The cursor of a JServerResultSet is initially positioned before the first row. An initial next() call is required to move the cursor to the first row.

Subsequent calls to next() add new rows; each should be preceded by set<Object>(int, <Object>) or setNull(int) calls to set column values for the row.

You can add any number of rows with next(). Once all rows have been added, call the done() method to indicate the end of the result set.

After the done() method finishes, the JServerResultSet is again positioned before the first row. The same JServerResultSet instance can be used to another result set based on the same metadata.

Implementations of the JServerResultSet interface may buffer rows as needed during consecutive next() calls before sending them to the client. The done() method should flush any buffered rows (and flush network buffers as well, if possible—the EAServer done() implementation flushes network buffers).

"Sending result sets with Java" in the *EAServer Programmer's Guide* summarizes the call sequences to send result sets and contains examples.

See also JContext.forwardResultSet(ResultSet)

JServerResultSet.done()

Description Indicates that all rows in a result set have been sent.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSet

```
public abstract void done()
   throws SQLException
```

Usage You must call the done() method to indicate that all rows in a result set have been sent.

JServerResultSet.findColumn(String)

Description Returns the index for the column that has the specified name.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSet

```
public abstract int findColumn( String columnName)
   throws SQLException
```

Parameters

columnName
The name of the column of interest.

Return value

The index of the column whose name matches the supplied name. Throws a SQLException if no column has a matching name. The index of the first column is 1.

See also

JServerResultSetMetaData.setColumnName(int, String)

JServerResultSet.getMetaData()

Description

Returns a java.sql.ResultSetMetaData object that describes the rows in a result set. The metadata includes the number of columns, the datatype of each column, and other details about each column, such as whether values can be NULL.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSet

```
public abstract ResultSetMetaData getMetaData()
  throws SQLException
```

Return value

A java.sql.ResultSetMetaData object that describes the rows in a result set.

Usage

A JServerResultSet object's metadata is determined when the object is constructed by calling createServerResultSetMetaData(). The metadata cannot be changed afterwards.

See also

java.sql.ResultSetMetaData, createServerResultSetMetaData(), createServerResultSet(JServerResultSetMetaData), java.sql.ResultSet.getMetaData()

JServerResultSet.next()

Description

Sends a row to the client.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSet

```
public abstract boolean next() throws SQLException
```

Return value

`true` if the row was successfully created, `false` otherwise.

Usage

The cursor of a JServerResultSet object is positioned before the first row when the object is constructed. An initial next() call is required to move the cursor to the first row. A done() call repositions the cursor before the first row.

After the first next() call, subsequent calls to next() add new rows; each should be preceded by set<Object>(int, <Object>) or setNull(int) calls to set column values for the row.

Any number of rows can be sent with next(). Once all rows have been sent, the done() method must be called to indicate the end of the result set.

See also

done(), ResultSet.next()

JServerResultSet.setBigDecimal(int, BigDecimal, int)

Description

Specifies a non-NULL value for a java.math.BigDecimal column.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSet

```
public abstract void setBigDecimal
   (int columnIndex,
    BigDecimal columnValue,
   int scale) throws SQLException
```

Parameters

columnIndex

The index of the column whose value is being set. The first column is 1.

columnValue

A java.math.BigDecimal value.

scale

The scale of the value. The scale specifies the number of decimal digits to the right of the decimal point.

Usage

Use setBigDecimal methods to specify values for non-NULL java.math.BigDecimal column values. If a column's value is NULL, call setNull(int).

You can set values for columns within a row in any order.

See also

ResultSet.getBigDecimal(int, int)

JServerResultSet.setCurrency(int, long)

Description

Specifies a non-NULL value for a column that represents a cash value.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSet

```
public abstract void setCurrency
   (int columnIndex,
   long columnValue)
   throws SQLException
```

Parameters

columnIndex

The index of the column whose value is being set. The first column is 1.

columnValue

The column's value, expressed as the number of one-ten-thousandths of a cash unit. In other words, *columnValue* represents the cash value:

```
columnValue/10000
```

Usage

You must call setCurrency to specify values for columns that represent a cash value. The result set's metadata specifies whether a column represents a cash value (ResultSetMetaData.isCurrency(int) returns true for the column).

setCurrency throws a SQLException if the column does not represent a cash value.

See also

ResultSet.getBigDecimal(int, int), ResultSetMetaData.isCurrency(int), JServerResultSetMetaData.setCurrency(int, boolean)

JServerResultSet.setNull(int)

Description

Specifies that a column in the current row has value NULL.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSet

```
public abstract void setNull(int columnIndex)
   throws SQLException
```

Parameters

columnIndex

The index of the column whose value is being set. The first column is 1.

Usage

An exception is thrown if the ResultSet object's metadata does not allow NULL values for the column.

See also JServerResultSetMetaData.setNullable(int, int), JServerResultSet.getMetaData(), ResultSet.wasNull()

JServerResultSet.set<Object>(int, <Object>)

Description Specifies a non-NULL value for a column in the current row.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSet

```
public abstract void setASCIIStream
   (int columnIndex, java.io.InputStream columnValue)
   throws SQLException, IOException

public abstract void setBinaryStream
   (int columnIndex, java.io.InputStream columnValue)
   throws SQLException, IOException

public abstract void setBoolean
   (int columnIndex, boolean columnValue)
   throws SQLException

public abstract void setByte
   (int columnIndex, byte columnValue)
   throws SQLException

public abstract void setDouble
   (int columnIndex, double columnValue)
   throws SQLException

public abstract void setDouble
   (int columnIndex, double columnValue)
   throws SQLException

public abstract void setFloat
   (int columnIndex, float columnValue)
   throws SQLException

public abstract void setInt
   (int columnIndex, int columnValue)
   throws SQLException

public abstract void setShort
   (int columnIndex, short columnValue)
   throws SQLException

public abstract void setString
   (int columnIndex, java.lang.String columnValue)
   throws SQLException
```

```
public abstract void setTimestamp
  (int columnIndex, java.sql.Timestamp columnValue)
  throws SQLException
```

Parameters

columnIndex
The index of the column whose value is being set. The first column is 1.

columnValue
An object of the appropriate type that contains the value for the column. The object type must match the column type that was specified by JServerResultSetMetaData.setColumnType(int, int) for the result set's metadata. Table 1-1 on page 41 lists type mappings.

Usage

Use the set<Object> methods to specify values for non-NULL column values. If a column's value is NULL, call setNull(int).

You can set values for columns within a row in any order.

See also

JServerResultSetMetaData.setColumnType(int, int), setBigDecimal(int, BigDecimal, int), java.sql.ResultSet

jaguar.sql.JServerResultSetMetaData interface

Description

```
package com.sybase.jaguar.sql;
public interface JServerResultSetMetaData
  extends ResultSetMetaData
```

Provides methods to describe a result set's metadata. Metadata specifies the number of columns in each row as well as the datatype, format, nullability, and so forth for each column.

Constructors

The JContext.createServerResultSetMetaData() method returns a class instance that implements this interface.

Methods

- setAutoIncrement(int, boolean) – (Not yet supported.) Specifies whether a column has the auto-increment property.
- setCaseSensitive(int, String) – (Not yet supported.) Specifies whether a column's values are case-sensitive.
- setCatalogName(int, String) – (Not yet supported.) Specifies the name of the column's catalog (database).
- setColumnCount(int) – Specifies the number of columns that will be sent in result-set rows.

- setColumnDisplaySize(int, int) – Specifies the column's normal maximum width in characters.
- setColumnLabel(int, String) – Recommends a display title for the column.
- setColumnName(int, String) – Specifies the column's name.
- setColumnType(int, int) – Specifies the column's SQL (java.sql.Types) datatype.
- setColumnTypeName(int, String) – (Not yet supported.) Specifies a column's data-source-specific type name.
- setCurrency(int, boolean) – Specifies whether the column represents a cash value.
- setNullable(int, int) – Specifies whether column values can be null.
- setPrecision(int, int) – Specifies the column's precision. The precision equals the number of decimal digits in a value.
- setScale(int, int) – Specifies the column's scale. The scale equals the number of decimal digits to the right of the decimal point.
- setSchemaName(int, String) – (Not yet supported.) Specifies the schema name of the column's table.
- setSearchable(int, boolean) – (Not yet supported.) Specifies whether a column can be used in a SQL where clause.
- setSigned(int, boolean) – (Not yet supported.) Specifies whether the column represents a signed number.
- setTableName(int, String) – (Not yet supported.) Specifies the name of the table that contains the column.

Note
The current version does not support some interface methods. The list above indicates the methods that are not yet supported. These methods throw a JException with a "Unsupported Functionality" message.

Usage

JServerResultSetMetaData provides set methods that correspond to the get methods defined in java.sql.ResultSetMetaData. Since JServerResultSetMetaData extends ResultSetMetaData, you can call the get methods directly on a JServerResultSetMetaData object.

You can use an initialized JServerResultSetMetaData object to create one or more JServerResultSet objects by calling JContext.createServerResultSet(JServerResultSetMetaData).

"Sending result sets with Java" in the *EAServer Programmer's Guide* summarizes the call sequences to send result sets and contains an example.

See also java.sql.ResultSetMetaData

JServerResultSetMetaData.setColumnCount(int)

Description Specifies the number of columns that will be sent in result-set rows.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSetMetaData

```
public abstract void setColumnCount(int columnCount)
  throws SQLException
```

Parameters *columnCount*
The number of columns.

Usage You must call setColumnCount() before you can call any other methods to describe an individual column's metadata. Once the number of columns is specified, it cannot be changed without discarding any column descriptions that you have set. That is, if you call setColumnCount() again, you must reset each column's metadata.

See also ResultSetMetaData.getColumnCount()

JServerResultSetMetaData.setColumnDisplaySize(int, int)

Description Specifies the column's normal maximum width in characters.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSetMetaData

```
public abstract void setColumnDisplaySize
  (int columnIndx, int size)
  throws SQLException
```

Parameters *columnIndex*
The index of the column. The first column has index 1.

size
The maximum width in characters.

Usage

setColumnDisplaySize determines the maximum length of variable length columns (CHAR, VARCHAR, LONGVARCHAR, BINARY, VARBINARY, LONGVARBINARY).

If you do not call setColumnDisplaySize to set a default display size, the implementation-specific default is used. To avoid excessive memory allocation, you must explicitly set the display size. In particular, the default display sizes for LONGVARCHAR and LONGVARBINARY columns can be larger than a Gigabyte.

See also

ResultSetMetaData.getColumnDisplaySize(int)

JServerResultSetMetaData.setColumnLabel(int, String)

Description

Recommends a display title for the column.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSetMetaData

```
public abstract void setColumnLabel
  (int columnIndex, String label)
  throws SQLException
```

Parameters

columnIndex

The index of the column. The first column has index 1.

label

The recommended display title. The default is the column name specified with setColumnName(int, String).

See also

ResultSetMetaData.getColumnLabel(int), setColumnName(int, String)

JServerResultSetMetaData.setColumnName(int, String)

Description

Specifies the column's name.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSetMetaData

```
public abstract void setColumnName
  (int columnIndex, String columnName)
  throws SQLException
```

Parameters

columnIndex
The index of the column. The first column has index 1.

columnName
The name of the column. The default is "" (0-length string).

See also

ResultSetMetaData.getColumnName(int)

JServerResultSetMetaData.setColumnType(int, int)

Description

Specifies the column's SQL (java.sql.Types) datatype.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSetMetaData

```
public abstract void setColumnType
  (int columnIndex, int SQLType)
  throws SQLException
```

Parameters

columnIndex
The index of the column. The first column has index 1.

SQLType
A symbolic constant that indicates the column's Java datatype. Constants are defined statically in the class java.sql.Types. The table below lists the supported java.sql.Types and lists, for each type, the corresponding Java type and the JServerResultSet.set<Object>(int, <Object>) method that must be called to set values for the column.

Table 1-1: Mapping type constants to Java types and setXXX methods

java.sql.Types constant	Java datatype	JServerResultSet method to set values
BINARY	byte[]	setBinaryStream or setBytes
BIT	boolean	setBoolean
CHAR	java.lang.String	setASCIIStream or setString
DECIMAL	java.math.BigDecimal	setBigDecimal
DOUBLE	double	setDouble
FLOAT	double	setDouble
INTEGER	int	setInt
LONGVARBINARY	java.io.InputStream or byte[]	setBinaryStream or setBytes
LONGVARCHAR	String	setASCIIStream or setString
NUMERIC	java.math.BigDecimal	setBigDecimal
REAL	float	setFloat
SMALLINT	short	setShort
TIMESTAMP	java.sql.Timestamp	setTimestamp
TINYINT	byte	setByte
VARCHAR	java.lang.String	setString
VARBINARY	byte[]	setBytes

Note
java.sql.Types.OTHER and java.sql.Types.BIGINT are not supported.

Usage

setColumnType(int, int) specifies the datatype for a column. There is no default. For java.math.BigDecimal columns, you must also call setPrecision(int, int) and setScale(int, int) to specify the column's precision and scale, respectively.

For columns that represent cash values, you must use JServerResultSet.setCurrency(int, long) to set values for the column.

See also

java.sql.Types, ResultSetMetaData.getColumnType(int), setPrecision(int, int), setScale(int, int)

JServerResultSetMetaData.setCurrency(int, boolean)

Description

Specifies whether the column represents a cash value.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSetMetaData

```
public abstract void
 setCurrency
   (int columnIndex, boolean property)
   throws SQLException
```

Parameters

columnIndex

The index of the column. The first column has index 1.

property

`true` if the column represents a cash value, `false` otherwise. The default is `false`.

See also

ResultSetMetaData.isCurrency(int)

JServerResultSetMetaData.setNullable(int, int)

Description

Specifies whether column values can be null.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSetMetaData

```
public abstract void setNullable
   (int columnIndex, int property)
   throws SQLException
```

Parameters

columnIndex

The index of the column. The first column has index 1.

property

A symbolic constant that takes the following values:

Value	To indicate
columnNullable	Values for the column can be null.
columnNoNulls	Values for the column cannot be null.
columnNullableUnknown	Nullability of the column is not known.

The default is columnNullableUnknown.

See also

JServerResultSet.setNull(int), ResultSetMetaData.isNullable(int)

JServerResultSetMetaData.setPrecision(int, int)

Description

Specifies the column's precision. The precision equals the number of decimal digits in a value.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSetMetaData

```
public abstract void setPrecision
   (int columnIndex, int precision)
   throws SQLException
```

Parameters

columnIndex

The index of the column. The first column has index 1.

precision

The precision of the column. The default is 0.

Usage

This method applies to java.math.BigDecimal columns only.

See also

ResultSetMetaData.getPrecision(int), setScale(int, int)

JServerResultSetMetaData.setScale(int, int)

Description

Specifies the column's scale. The scale equals the number of decimal digits to the right of the decimal point.

Syntax

Package	com.sybase.jaguar.sql
Interface	JServerResultSetMetaData

```
public abstract void setScale
   (int columnIndex, int scale)
   throws SQLException
```

Parameters

columnIndex

The index of the column. The first column has index 1.

scale

The scale for the column. The default is 0.

Usage

This method applies to java.math.BigDecimal columns only.

See also

ResultSetMetaData.getScale(int), setPrecision(int, int)

jaguar.util.JException class

Description

```
package com.sybase.jaguar.util;
public class JException
  extends Exception
```

JException is the generic exception that is thrown by methods in the EAServer classes or in generated client stub classes.

Constructors — Same as java.lang.Exception.

Methods — Same as java.lang.Exception.

See also — JConnectionNotFoundException, java.sql.SQLException

jaguar.util.<object>Holder class

Description

```
package com.sybase.jaguar.util;
public class <object>Holder extends Object
```

For components that use the Jaguar-JDBC type mappings, holder classes are used to pass INOUT parameters to component method calls. Each holder class has a *value* field that contains instances of a specific object or base Java type.

Additional holder classes are defined in packages com.sybase.jaguar.util.jdbc102 and com.sybase.jaguar.util.jdbc11.

com.sybase.jaguar.util holder classes are summarized in the Table 1-2.

Table 1-2: Holder classes

Holder class	Datatype for value field	Default for value
`BooleanHolder`	`boolean`	false
`ByteHolder`	`byte`	0
`BytesHolder`	`byte[]`	null
`CharHolder`	`char`	\u0000 (null character)
`FloatHolder`	`float`	0.0
`DoubleHolder`	`double`	0.0
`IntegerHolder`	`int`	0
`LongHolder`	`long`	0
`ShortHolder`	`short`	0
`StringHolder`	`java.lang.String`	null

Warning! Null parameter values are not supported. For StringHolder or BytesHolder parameters, use the constructor that takes an initial value, or set the value field explicitly.

Constructors

<object>Holder()

Default constructor that assigns the default value specified in Table 1-2 on page 45.

<object>Holder(<object> initialValue)

Constructor that takes an initial value specified as *initialValue. initialValue* is an instance of the appropriate datatype as specified in Table 1-2 on page 45.

Fields

value

The current value contained by the holder object. Table 1-2 on page 45 lists the datatypes and default values for the *value* field.

Usage

Java component methods on the server receive INOUT parameters as a holder object. The method should set the value field of each holder object before returning.

The examples below illustrate how to construct and use holder objects:

- Each holder object has a default constructor that takes no arguments. For example:

```
StringHolder str_holder = new StringHolder();
IntegerHolder int_holder = new IntegerHolder();
```

- Each holder object has an additional constructor that takes an initial value as a parameter. For example:

```
StringHolder str_holder =
                  new StringHolder("hello");
IntegerHolder int_holder = new IntegerHolder(43);

float f = 3.141;
FloatHolder float_holder = new FloatHolder(f);
```

- Each holder object has a value member that allows access to the base object value. For example:

```
IntegerHolder i_hold = new IntegerHolder();
System.out.println(
  "IntegerHolder default value is:"
  + i_hold.value);
```

See also jaguar.util.jdbc102.<object>Holder class, jaguar.util.jdbc11.<object>Holder class

jaguar.util.jdbc102.<object>Holder class

Description

package com.sybase.jaguar.util.jdbc102;
public class <object>Holder extends Object

The com.sybase.jaguar.util.jdbc102 holder classes are used to pass jdbc.sql and jdbc.math objects as INOUT parameters.

For code that runs in a JDK 1.0.2 virtual machine, use these imports:

```
import jdbc.sql.*;
import jdbc.math.*;
import com.sybase.jaguar.util.jdbc102.*;
```

The jdbc.sql package contains classes that are equivalent to JDK 1.1 java.sql classes that have the same name. The jdbc.math package contains classes that are equivalent to JDK 1.1 java.math classes that have the same name. For details, see the JDK 1.1 documentation of the java.math and java.sql packages.

The holder classes for JDK 1.0.2 are summarized in Table 1-3:

Table 1-3: Holder classes for use with JDK 1.0.2

Holder class	Datatype for value field	Default for value
`BigDecimalHolder`	`jdbc.math.BigDecimal`	null
`DateHolder`	`jdbc.sql.Date`	null
`TimeHolder`	`jdbc.sql.Time`	null
`TimestampHolder`	`jdbc.sql.Time`	null

Warning! Null parameter values are not supported. Use the constructor that takes an initial value, or set the value field explicitly.

Constructors

<object>Holder()

Default constructor that assigns the default value specified in Table 1-3.

<object>Holder(<object> initialValue)

Constructor that takes an initial value specified as *initialValue. initialValue* is an instance of the appropriate datatype as specified in Table 1-3.

Fields

value
The current value contained by the holder object.Table 1-3 lists the datatypes and default values for the *value* field.

See also

jaguar.util.<object>Holder class, jaguar.util.jdbc11.<object>Holder class

jaguar.util.jdbc11.<object>Holder class

Description

package com.sybase.jaguar.util.jdbc11;
public class <object>Holder extends Object

The com.sybase.jaguar.util.jdbc11 holder classes are used to pass java.sql and java.math objects as INOUT parameters. Use these classes in code that runs in a JDK 1.1 or later virtual machine. See jaguar.util.jdbc102.<object>Holder class for similar classes that are compatible with JDK 1.0.2.

For code that will be run in a JDK 1.1 or later virtual machine, use these imports:

```
import java.sql.*;
import java.math.*;
import com.sybase.jaguar.util.jdbc11.*;
```

The holder classes for JDK 1.1 are summarized in Table 1-4:

Table 1-4: Holder for use with JDK 1.1

Holder class	Datatype for value field	Default for value
`BigDecimalHolder`	`java.math.BigDecimal`	null
`DateHolder`	`java.sql.Date`	null
`TimeHolder`	`java.sql.Time`	null
`TimestampHolder`	`java.sql.Timestamp`	null

Warning! Null parameter values are not supported. Use the constructor that takes an initial value, or set the value field explicitly.

Constructors

<object>Holder()

Default constructor that assigns the default value specified in Table 1-4.

<object>Holder(<object> initialValue)

Constructor that takes an initial value specified as *initialValue. initialValue* is an instance of the appropriate datatype as specified in Table 1-4.

Fields

value

The current value contained by the holder object.Table 1-4 lists the datatypes and default values for the *value* field.

See also

jaguar.util.<object>Holder class, jaguar.util.jdbc102.<object>Holder class

CHAPTER 2

ActiveX C++ Interface Reference

This chapter documents the custom interfaces for the EAServer server-side ActiveX objects. These interfaces are defined in the C++ header file *jagctx.h*. Some objects also provide an IDispatch interface that allows the object to be used in ActiveX automation IDEs such as PowerBuilder. Chapter 3, "ActiveX IDispatch Interface Reference," provides reference pages for the IDispatch interfaces.

Header files and link libraries

All the interfaces documented here are defined in *jagctx.h*. Link information is in *libjdispatch.lib*. You must include *jagctx.h* in source code that uses these interfaces, and link *libjdispatch.lib* when building the DLL. Add the EAServer *include* subdirectory to your compiler's header-file search path. Add the EAServer *lib* subdirectory to your compiler's library-file search path.

To use the ISharedPropertyGroupManager, ISharedPropertyGroup, and ISharedProperty interfaces, you must include *JagSharedProp.h*. Additionally, you must include *JagSharedProp_i.c* in one—and only one—source file for your component DLL. *JagSharedProp.h* contains interface definitions for the documented interfaces. *JagSharedProp_i.c* declares symbols that are required by the ActiveX CoCreateInstance routine. (CoCreateInstance is called to create ISharedPropertyGroupManager interface pointers.)

Warning! If you include *JagSharedProp_i.c* in more than one source file for your component DLL, you will get duplicate-symbol errors when linking.

List of interfaces

- GetObjectContext routine – Retrieves the object context interface that is associated with your component instance.
- IJagServer interface – Provides utility methods for use in EAServer ActiveX components.
- IJagServerResults interface – Provides methods to send rows to a EAServer client application.
- IObjectContext interface – Provides methods that allow your component to influence the transaction outcome. Nontransactional components can call the IObjectContext methods to cause early deactivation of an instance.
- IObjectControl interface – Allows components to support EAServer's instance pooling model. The component dispatcher calls the IObjectControl methods to indicate transitions in the lifecycle of an ActiveX component.
- ISharedProperty interface – Represents a property value that is shared among all ActiveX component instances in a EAServer package.
- ISharedPropertyGroup interface – Represents a group of properties that are shared by all ActiveX components in a EAServer package. Contains methods to create, access, and destroy shared properties.
- ISharedPropertyGroupManager interface – Contains methods to create, access, and destroy shared property groups.

GetObjectContext routine

Description

Retrieves the object context interface that is associated with your component instance.

Syntax

```
#include <jagctx.h>

HRESULT GetObjectContext (
        IObjectContext ** ppInstCtx
);
```

Parameters

ppInstCtx

The address of an IObjectContext interface pointer.

Return value

Return value	To indicate
S_OK	Successful retrieval of the IObjectContext interface pointer.
E_INVALIDARG	*ppInstCtx* was NULL.
CONTEXT_E_NOCONTEXT	GetObjectContext was called from code that was not executing as part of a component method invocation. This can happen if you run your code outside of EAServer or if you call GetObjectContext from the component's constructor.

Usage

Call GetObjectContext to obtain an IObjectContext interface pointer.

GetObjectContext is defined in *mtx.h*; to call it, you must link *mtx.lib* with your component.

The IObjectContext interface is not available unless GetObjectContext is called from code that is executing in the context of a component method invocation.

The IObjectContext interface is not available in the component's class constructor.

An IObjectContext reference is not valid after an instance has been deactivated. If your component implements the IObjectControl interface, you can obtain an IObjectContext pointer in the Activate method and release it when Deactivate is called. Components that do not implement IObjectControl can obtain an IObjectContext pointer and release it in the destructor.

See also

IObjectContext interface

IJagServer interface

Description

Provides utility methods for use in EAServer ActiveX components.

Methods

- WriteLog – Writes a message to the server's log file.

Usage

To create an IJagServer interface pointer, use the ProgID, "Jaguar.JagServer.1". Call the OLE routines CLSIDfromProgID and CoCreateInstance. CoCreateInstance returns an interface pointer for a given ActiveX class ID string. CLSIDfromProgID obtains the class ID string that CoCreateInstance requires.

To use the IJagServer and IJagServerResults interfaces, you must include *JagAxWrap.h*. Additionally, you must include *JagAxWrap_i.c* in only one source file for your component DLL. *JagAxWrap.h* contains interface definitions for the documented interfaces. *JagAxWrap_i.c* declares symbols that are required by the ActiveX CoCreateInstance routine.

Warning! You will get duplicate-symbol link errors if you include *JagAxWrap_i.c* in more than one source file for your component DLL.

See also

Chapter 19, "Creating ActiveX Components," in the *EAServer Programmer's Guide*

IJagServer::WriteLog

Description

Writes a message to the server's log file.

Syntax

```
#include <JagAxWrap.h>

HRESULT IJagServer::WriteLog(
    VARIANT_BOOL useTimeStamp,
    BSTR message)
```

Parameters

useTimeStamp

`VARIANT_TRUE` if the current date and time should be prepended to the log message; `VARIANT_FALSE` otherwise.

message

A message to be written to the server's log file.

Return value

Return value	To indicate
S_OK	Successful execution.
E_OUTOFMEMORY	Out of memory.
E_FAIL	Failure. WriteLog fails if the log file cannot be opened or if *message* is NULL. If the log file cannot be opened, log messages are written to the server process' standard error device.

Usage

This method records a message in the server's log file.

By convention, errors that occur on the server are recorded in the log. Log messages should contain enough detail for an administrator or programmer to troubleshoot the cause of the error.

After recording error information in the log, you can also send a concise description of the error by raising an OLE automation exception.

For information on configuring the log file used by the server, see Chapter 3, "Creating and Configuring Servers," in the *EAServer System Administration Guide*.

When coding in C++, you can call the C routine JagLog instead of IJagServer::WriteLog. Calling the C routine avoids the overhead incurred by creating an IJagServer interface pointer.

Example

The following C++ code fragment creates an IJagServer interface pointer and calls WriteLog to log the message "Hello, logfile":

```
HRESULT     hr;
IJagServer *p_ijs;
CLSID       clsid_js;
BSTR        msg;

// Create an IJagServer interface pointer
hr = CLSIDFromProgID(L"Jaguar.JagServer.1", &clsid_js)
;
// ... deleted error checking ...

hr = CoCreateInstance(clsid_js, NULL,
                      CLSCTX_INPROC_SERVER,
                      IID_IJagServer,
                      (void**)&p_ijs);
// ... deleted error checking ...

msg = SysAllocString(L"Hello, logfile\n");
// ... deleted error checking ...
hr = p_ijs->WriteLog(VARIANT_TRUE, msg);
// ... deleted error checking ...
```

See also

Chapter 19, "Creating ActiveX Components," in the *EAServer Programmer's Guide*

JagLog in Chapter 5, "C Routines Reference."

IJagServerResults interface

Description

Provides methods to send rows to a EAServer client application.

Methods

- BeginResults – Begins the sequence of calls that sends a result set to the client.
- BindCol – Binds a program variable to a column in a result set.
- ColAttributes – Specifies additional metadata for a column to be sent in a result set.
- DescribeCol – Describes a result-set column.
- EndResults – Indicates that all rows in a result set have been sent.
- ResultsPassthrough – Forwards results from a remote database query to the client
- SendData – Sends one row in a result set.

Usage

To create an IJagServer interface pointer, use the ProgID, "EAServer.JagServerResults.1". Call the OLE routines CLSIDfromProgID and CoCreateInstance. CoCreateInstance returns an interface pointer for a given ActiveX class ID string. CLSIDfromProgID obtains the class ID string that CoCreateInstance requires.

To use the IJagServerResults and IJagServer interfaces, you must include *JagAxWrap.h*. Additionally, you must include *JagAxWrap_i.c* in only one source file for your component DLL. *JagAxWrap.h* contains interface definitions for the documented interfaces. *JagAxWrap_i.c* declares symbols that are required by the ActiveX CoCreateInstance routine.

Warning! You will get duplicate-symbol link errors if you include *JagAxWrap_i.c* in more than one source file for your component DLL.

See also

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServerResults::BeginResults

Description

Begins the sequence of calls that sends a result set to the client.

Syntax

```
#include <JagAxWrap.h>

HRESULT IJagServerResults::BeginResults(
    short numColumns
    )
```

Parameters

numColumns

The number of columns in the result set to be sent.

Return value

Return value	To indicate
S_OK	Successful execution
E_INVALIDARG	*numColumns* was not a positive number
E_OUTOFMEMORY	Out of memory
E_FAIL	Failure. Check the server's log file for information about the cause of failure

See also

DescribeCol

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServerResults::BindCol

Description

Binds a program variable to a column in a result set.

Syntax

```
#include <JagAxWrap.h>

HRESULT IJagServerResults::BindCol(
    short item,
    VARIANTARG sourceBuf,
    long maxBuflen,
    short *indicator)
```

Parameters

item

The column number. The first column is 1.

sourceBuf

A VARIANTARG structure that describes the C datatype of the variable that holds data values. The table below summarizes how to set the VARIANTARG fields. You must set the *vt* field to indicate the C type for the supplied column data, then use the indicated field to specify the address of another variable that holds column values. Subsequent calls to SendData read values from the variable at the indicated address; the address must remain valid until EndResults is called.

Table 2-1: VARIANTARG settings for BindCol

C datatype	vt field setting	Field that specifies bound variable address
SHORT *	VT_I2 | VT_BYREF	*piVal*
LONG *	VT_I4 | VT_BYREF	*plVal*
FLOAT *	VT_R4 | VT_BYREF	*pfltVal*
DOUBLE *	VT_R8 | VT_BYREF	*pdblVal*
VARIANT_BOOL *	VT_BOOL | VT_BYREF	*pbool*
DATE *	VT_DATE | VT_BYREF	*pdate*
SAFEARRAY *	VT_ARRAY | VT_UI1 | VT_BYREF	*pparray*
BSTR *	VT_BSTR | VT_BYREF	*pbstrVal*

Use BSTR for string values and SAFE_ARRAY for binary values. Decimal and currency values can be specified as string data (BSTR) or any other type that can be converted to a numeric fraction, such as SHORT, LONG, FLOAT or DOUBLE. “ActiveX to SQL Datatype conversion” on page 58 describes the supported conversions between SQL and ActiveX datatypes.

BindCol copies the structure contents before returning, consequently:

- You can use one VARIANTARG structure to set up binds for all columns in a result set, and
- Changes made to the structure after BindCol returns have no effect.

maxBuflen

For string or binary values, the maximum length for column values that can be sent. Ignored for other datatypes.

indicator

The address of a variable that acts as a null-indicator for column values. Subsequent calls to SendData read the null-indicator value to determine whether a null value should be sent to the client. Null-indicator values are as follows:

Value	To indicate
0	Column value is not null and must be read from the variable indicated by the *sourceBuf* `Variant` buffer.

Value	To indicate
-1	Column value is null.

The *indicator* reference must remain valid until EndResults is called.

Return value

Return value	To indicate
S_OK	Successful execution.
E_INVALIDARG	At least one parameter contained an invalid value. Check the server's log file for more information.
E_OUTOFMEMORY	Out of memory.
E_FAIL	Failure. Check the server's log file for information about the cause of failure.

Usage

BindCol associates a program variable with a column in a result set. When SendData is called to send a row, it reads the column value for the current row from the variable that is bound to the column.

ActiveX to SQL Datatype conversion

The *SQLDatatype* value passed to DescribeCol determines the datatype with which column values are sent over the network. If the program variable type does not map directly to the column's SQL datatype, SendData attempts to convert the value. The figure below shows the supported conversions between SQL datatypes and bind variable types. An X indicates a supported conversion.

SQL Datatype / Automatoin Datatype	BSTR	VARIANT BOOL	short	long	float	double	SAFEARRAY	DATE
"SQL_CHAR"	X	X	X	X	X	X	X	X
"SQL_VARCHAR"	X	X	X	X	X	X	X	X
"SQL_LONGVARCHAR"	X	X	X	X	X	X	X	X
"SQL_DECIMAL"	X	X	X	X	X	X	X	
"SQL_NUMERIC"	X	X	X	X	X	X	X	
"SQL_BIT"	X	X	X	X	X	X	X	
"SQL_TINYINT"	X	X	X	X	X	X	X	
"SQL_SMALLINT"	X	X	X	X	X	X	X	
"SQL_INTEGER"	X	X	X	X	X	X	X	
"SQL_REAL"	X	X	X	X	X	X	X	
"SQL_FLOAT"	X	X	X	X	X	X	X	
"SQL_DOUBLE"	X	X	X	X	X	X	X	
"SQL_BINARY"	X	X	X	X	X	X	X	
"SQL_VARBINARY"	X	X	X	X	X	X	X	
"SQL_LONGVARBINARY"	X	X	X	X	X	X	X	
"SQL_DATE"	X						X	X
"SQL_TIME"	X						X	
"SQL_TIMESTAMP"	X						X	X

See also

DescribeCol, EndResults, SendData

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServerResults::ColAttributes

Description

Specifies additional metadata for a column to be sent in a result set.

Syntax

```
#include <JagAxWrap.h>

HRESULT IJagServerResults::ColAttributes(
    short item,
    BSTR descType,
```

```
    VARIANTARG descBuf
)
```

Parameters

item
The column number. The first column is 1.

descType
A BSTR; must be initialized to “COLUMN_MONEY”.

descBuf
A VARIANTARG structure initialized to contain a VARIANT_BOOL. A value of TRUE means that the column represents a cash value.

Return value

Return value	To indicate
S_OK	Successful execution.
E_INVALIDARG	At least one parameter contained an invalid value. Check the server’s log file for more information.
E_FAIL	Failure. Check the server’s log file for information about the cause of failure.

Usage

If a column in a result set represents a cash value, you must call ColAttributes to set the “COLUMN_MONEY” attribute to TRUE. This attribute defaults to FALSE.

See also

DescribeCol

Chapter 25, “Sending Result Sets,” in the *EAServer Programmer’s Guide*

IJagServerResults::DescribeCol

Description

Describes a result-set column.

Syntax

```
#include <JagAxWrap.h>

HRESULT IJagServerResults::DescribeCol(
    short item,
    BSTR columnName,
    BSTR SQLDatatype,
    long columnSize,
    long precision,
    short scale,
    VARIANT_BOOL nullable
)
```

Parameters

item
The column number. The first column is 1.

columnName

A BSTR containing the column's name.

SQLDatatype

A BSTR containing the name of the column's SQL datatype. This value determines the datatype of values sent to the client. Values are specified in a buffer that is bound to the column with BindCol. The following table lists the datatype strings. See BindCol for details on how to bind column values.

Table 2-2: SQL datatypes for DescribeCol

SQL datatype string	Description
"SQL_BIT"	Boolean. A single bit of data.
"SQL_TINYINT"	A 1-byte integer.
"SQL_SMALLINT"	A 2-byte integer.
"SQL_INTEGER"	A 4-byte integer.
"SQL_REAL"	A 4-byte floating point number.
"SQL_FLOAT"	An 8-byte floating point number.
"SQL_DOUBLE"	Same as "SQL_FLOAT".
"SQL_NUMERIC"	A fixed-point fractional decimal number.
"SQL_DECIMAL"	Same as "SQL_DECIMAL".
"SQL_CHAR"	A string of characters. Values do not vary in length, and the specified length (*columnSize*) must be less than 256.
"SQL_VARCHAR"	A string of characters. Values may vary in length and have maximum length specified by *columnSize*. *columnSize* must be < 256.
"SQL_LONGVARCHAR"	A string of characters. Values may vary in length and have maximum length specified by *columnSize*. *columnSize* is constrained by available memory.
"SQL_DATE"	An ODBC date value.
"SQL_TIME"	An ODBC time value.
"SQL_TIMESTAMP"	An ODBC timestamp value.
"SQL_BINARY"	An array of bytes that does not vary in length. The specified length (*columnSize*) must be less than 256.
"SQL_VARBINARY"	An array of bytes that can vary in length. The specified maximum length (*columnSize*) must be less than 256.
"SQL_LONGVARBINARY"	An array of bytes that can vary in length. The specified maximum length (*columnSize*) is constrained by available memory.

columnSize
For character or binary columns, the maximum length for column values.

precision
The precision of column values. For "SQL_NUMERIC" or "SQL_DECIMAL" columns, *precision* indicates the maximum number of decimal digits that a value may have. For other datatypes, *precision* is ignored.

scale
The scale for column values. For "SQL_NUMERIC" or "SQL_DECIMAL" columns, *scale* indicates the number of decimal digits to the right of the decimal point. For other datatypes, *scale* is ignored.

nullable
VARIANT_TRUE if the column may have null values.

Return value

Return value	To indicate
S_OK	Successful execution.
E_INVALIDARG	At least one parameter contained an invalid value. Check the server's log file for more information.
E_OUTOFMEMORY	Out of memory.
E_FAIL	Failure. Check the server's log file for more information.

Usage

DescribeCol describes the datatype, name, and format of a result column. The ColAttributes method specifies additional metadata.

See also

BindCol, ColAttributes

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServerResults::EndResults

Description

Indicates that all rows in a result set have been sent.

Syntax

```
#include <JagAxWrap.h>

HRESULT IJagServerResults::EndResults(long rowCount);
```

Parameters

rowCount
The number of rows that were sent.

Return value

Return value	To indicate
S_OK	Successful execution.
E_FAIL	Failure. EndResults fails if *rowCount* was negative. Check the server's log file for information about the cause of failure.

See also

BeginResults

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServerResults::ResultsPassthrough

Description

Forwards results from a remote database query to the client.

Syntax

```
#include <JagAxWrap.h>

HRESULT IJagServerResults::ResultsPassthrough(
    BSTR conlibName,
    VARIANTARG *conlibPtr,
    BSTR pthruType,
    long *pInfo
)
```

Parameters

conlibName

A BSTR with one of the following values:

- "ODBC" to indicate that *conlibPtr* contains the address of an ODBC HSTMT control structure.
- "CTLIB" to indicate that *conlibPtr* contains the address of a Client-Library CS_COMMAND control structure.

conlibPtr

A VARIANTARG structure containing the ODBC HSTMT or Client-Library CS_COMMAND control structure. Set the VARIANTARG *vt* field to VT_BYREF and the *byref* field to the address of the control structure.

When using ODBC, the HSTMT must be in a state that allows SQLFetch to be called without error.

When using Client-Library, the CS_COMMAND structure must be in a state that allows ct_results to be called without error.

pthruType

A BSTR with one of the following values:

- "CURRENT_RESULTS" to indicate that only the current result set should be forwarded to the client. When using this option, you must ensure that all result sets are processed. You can call ResultsPassthrough in a loop (see examples in "Comments" below). You can also retrieve or cancel subsequent result sets by directly calling ODBC or Client-Library routines.
- "ALL_RESULTS" to indicate that all result sets should be forwarded to the client.

pInfo

Pass as NULL if using ODBC or if using the "ALL_RESULTS" option to forward all results with one call.

When forwarding individual Client-Library result sets, pass the address of long variable as *pInfo*. ResultsPassthrough sets the *pInfo* variable to specify whether all results have been retrieved from the CS_COMMAND structure, as follows:

*pInfo value	To indicate
`NoInfo` (0)	More results remain to be processed.
`NoMoreResults` (1)	All results have been processed.

Return value

Return value	To indicate
S_OK	Successful execution.
E_INVALIDARG	At least one parameter contained an invalid value. Check the server's log file for more information.
E_FAIL	Failure. Check the server's log file for information about the cause of failure.

Usage

ResultsPassthrough forwards ODBC or Client-Library result sets to the client.

All results from a query can be forwarded with one call using the "ALL_RESULTS" option for the *pthruType* parameter. To forward single result sets, use the "CURRENT_RESULTS" option.

When using the JAG_PTHRU_ALL_RESULTS option with Client-Library, any result type other than row results (CS_ROW_RESULTS) causes ResultsPassthrough to fail.

When forwarding single result sets, you must ensure that you retrieve or cancel all results. The sections below describe the loop algorithms for forwarding individual result sets.

Forwarding Individual Result Sets with Client-Library

When using the "CURRENT_RESULTS" option with Client-Library, call ResultsPassthrough in place of calling ct_results. You must pass the address of a LONG as the *pInfo* variable. If this variable is 1 when ResultsPassthrough returns, no more results are available from the CS_COMMAND structure. The code fragment below illustrates how ResultsPassthrough can be called in a loop:

```
HRESULT         hr;
CS_RETCODE      retcode;
CS_CHAR         *sqlCmd =
    "select * from titles select * from authors"
CS_COMMAND      *cmd;
VARIANT         theVariant;
long            info;
IJagServerResults  *pResApis;

// Deleted code which retreived the pointer to the
// JagServerResults interface.
// Also, deleted the code which did CT-Lib
// initialization, connected to the SQL Server,
// and allocated the CS_COMMAND structure.

retcode = ct_command(cmd, CS_LANG_CMD, sqlCmd,
              CS_NULLTERM, CS_UNUSED);
if (retcode != CS_SUCCEED)
{
    // handle failure
}
retcode = ct_send(cmd);
if (retcode != CS_SUCCEED)
{
    // handle failure
}

theVariant.vt = VT_BYREF;
theVariant.byref = cmd;
while ((hr = pResApis-
>ResultsPassthrough("CTLIB", theVariant,
                 "CURRENT_RESULTS", &info)) == S_OK)
{
    if (info == NoMoreResults)
```

```
    {
        break;
    }
}
if (hr != S_OK)
{
    // handle failure
}
```

Forwarding Individual Result Sets with ODBC

When using the "CURRENT_RESULTS" option with ODBC, call ResultsPassthrough before calling SQLMoreResults, instead of the usual SQLFetch row processing. The code fragment below illustrates how ResultsPassthrough and SQLMoreResults can be called in a loop to forward all result sets to the client.

```
HRESULT     hr;
RETCODE     odbcRet;
CS_CHAR     *sqlCmd =
        "select * from titles select * from authors"
HSTMT       hstmt;
VARIANT     theVariant;
long        info;
IJagServerResults       *pResApis;

// Deleted code which retreived the pointer to the
// JagServerResults interface.
// Also, deleted the code which did ODBC initializatio
n,
// connected to the SQL Server, and allocated the HSTM
T.

odbcRet = SQLExecDirect(hstmt, (SQLCHAR *)sqlCmd, SQL_
NTS);
if (odbcRet != SQL_SUCCESS)
{
    // handle failure
}

theVariant.vt = VT_BYREF;
theVariant.byref = &hstmt;
do
{
    hr = pResApis-
>ResultsPassthrough("ODBC", theVariant,
                "CURRENT_RESULTS", &info);
```

```
        if (hr != S_OK)
        {
            // handle failure
        }
    } while (SQLMoreResults == SQL_SUCCESS);
    if (odbcRet != SQL_NO_DATA_FOUND)
    {
        // handle failure
    }
```

See also Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServerResults::SendData

Description Sends one row in a result set.

Syntax

```
#include <JagAxWrap.h>

HRESULT IJagServerResults::SendData(void);
```

Return value

Return value	To indicate
S_OK	Successful execution.
E_INVALIDARG	At least one parameter contained an invalid value. Check the server's log file for more information.
E_FAIL	Failure. Check the server's log file for information about the cause of failure.

Usage After you have described columns with DescribeCol and bound program variables to supply column data, call SendData to send each row of data.

See also DescribeCol, BindCol, EndResults

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IObjectContext interface

Description Provides methods that allow your component to influence the transaction outcome. Nontransactional components can call the IObjectContext methods to cause early deactivation of an instance.

Methods

- DisableCommit – Indicates that the current transaction cannot be committed because the component's work has not been completed; the instance remains active after the current method returns.
- EnableCommit – Indicates that the component should not be deactivated after the current method invocation; allow the current transaction to be committed if the component instance is deactivated.
- IsInTransaction – Determines whether the current method is executing in a transaction.
- IsSecurityEnabled – Determines whether login security and component authorization are enabled for the server.
- SetAbort – Indicates that the component cannot complete its work for the current transaction and that the transaction should be rolled back. The component instance will be deactivated when the method returns.
- SetComplete – Indicates that the component's work for the current transaction was successfully finished and that this component instance should be deactivated when the method returns.

The following methods are not supported and always return an HRESULT status of DISP_E_NOTIMPLEMENTED:

- CreateInstance
- IsCallerInRole
- SafeRef

Usage

The IObjectContext interface contains methods that allow your component to influence the transaction outcome.

Call the GetObjectContext routine to obtain an IObjectContext interface pointer.

See also

GetObjectContext routine, IObjectControl interface

IObjectContext::DisableCommit

Description

Indicates that the current transaction cannot be committed because the component's work has not been completed; the instance remains active after the current method returns.

Syntax

```
#include <jagctx.h>

HRESULT IObjectContext::DisableCommit (void);
```

Return value

Return value	To indicate
S_OK	Successfully set transactional state.
CONTEXT_E_NOCONTEXT	DisableCommit was called from code that was not executing as part of a component method invocation.

Usage

DisableCommit specifies that the component instance should not be automatically deactivated after the current method completes. If the instance is deactivated before the next method invocation, the current transaction is rolled back.

When a method calls DisableCommit, the component instance is not deactivated until one of the following happens:

- The component's stub is destroyed explicitly by the client.
- The client disconnects without explicitly destroying the stub (the current transaction is always rolled back in this case).
- The component instance calls IObjectContext::SetComplete or IObjectContext::SetAbort during a subsequent method invocation.

EnableCommit and DisableCommit allow a component maintain state between method calls. If a component is not transactional, these two methods have the same effect: both prevent immediate deactivation of the component.

If a method calls none of DisableCommit, EnableCommit, SetAbort, or SetComplete, the default behavior is that of EnableCommit.

See also

EnableCommit, SetAbort, SetComplete

Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide*

IObjectContext::EnableCommit

Description

Indicates that the component should not be deactivated after the current method invocation; allow the current transaction to be committed if the component instance is deactivated.

Syntax

```
#include <jagctx.h>

HRESULT IObjectContext::EnableCommit (void);
```

Return value

Return value	To indicate
S_OK	Successfully set transactional state.
CONTEXT_E_NOCONTEXT	EnableCommit was called from code that was not executing as part of a component method invocation.

Usage

EnableCommit specifies that the component instance should not be automatically deactivated after the current method completes. If the instance is deactivated before the next method invocation, the current transaction is committed.

When a method calls EnableCommit, the component instance is not deactivated until one of the following happens:

- The transaction times out or the client's instance reference expires. In either case, the current transaction is rolled back.
- The transaction's root component calls SetComplete or SetAbort.
- The component instance calls SetComplete or SetAbort during a subsequent method invocation.

EnableCommit and DisableCommit allow a component maintain state between method calls. If a component is not transactional, these two methods have the same effect: both prevent immediate deactivation of the component.

If a method calls none of DisableCommit, EnableCommit, SetAbort, or SetComplete, the default behavior is that of EnableCommit.

See also

DisableCommit, SetAbort, SetComplete

Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide*

IObjectContext::IsInTransaction

Description

Determines whether the current method is executing in a transaction.

Syntax

```
#include <jagctx.h>

BOOL IObjectContext::IsInTransaction (void);
```

Return value

Return value	To indicate
TRUE	The current method invocation is executing within a EAServer transaction.

Return value	To indicate
FALSE	The current method invocation is not executing within a EAServer transaction.

Usage

Methods can call IsInTransaction to determine whether they are executing within a transaction. Methods in components that are declared to be transactional always execute as part of a transaction.

See also

Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide*

IObjectContext::IsSecurityEnabled

Description

Determines whether login security and component authorization are enabled for the server.

Syntax

```
#include <jagctx.h>

BOOL IObjectContext::IsSecurityEnabled (void);
```

Return value

Return value	To indicate
TRUE	Security is enabled.
FALSE	Security is not enabled.

Usage

By default, login security and component authorization are disabled for newly installed servers.

Note
In the current release, IsSecurityEnabled returns TRUE regardless of whether security has been enabled in the server configuration.

IObjectContext::SetAbort

Description

Indicates that the component cannot complete its work for the current transaction and that the transaction should be rolled back. The component instance will be deactivated when the method returns.

Syntax

```
#include <jagctx.h>

HRESULT IObjectContext::SetAbort (void);
```

Return value

Return value	To indicate
S_OK	Successfully set transactional state.
CONTEXT_E_NOCONTEXT	SetAbort was called from code that was not executing as part of a component method invocation.

Usage

SetAbort specifies that the component cannot complete its work for the current transaction. The transaction will be rolled back when the initiating component is deactivated.

If a component is not transactional, then SetAbort and SetComplete have the same effect: both cause the component instance to deactivate after the currently executing method returns.

If a method calls none of DisableCommit, EnableCommit, SetAbort, or SetComplete, the default behavior is that of EnableCommit.

See also

DisableCommit, EnableCommit, IsInTransaction, SetComplete

Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide*

IObjectContext::SetComplete

Description

Indicates that the component's work for the current transaction was successfully finished and that this component instance should be deactivated when the method returns.

Syntax

```
#include <jagctx.h>

HRESULT IObjectContext::SetComplete (void);
```

Return value

Return value	To indicate
S_OK	Successfully set transactional state.
CONTEXT_E_NOCONTEXT	SetComplete was called from code that was not executing as part of a component method invocation.

Usage

SetComplete specifies that the component has successfully completed its contribution to the current transaction. The component instance deactivates when control returns from the current component method invocation.

If the component instance is the initiator of the transaction (that is, it was instantiated directly by a base client), then EAServer attempts to commit the transaction. The transaction commits unless the commit is disallowed or vetoed; depending on the components that are participating, this can happen in any of the following ways:

- A participating C component has called JagDisallowCommit.
- A participating Java component throws an exception from its ServerBean.deactivate() method.
- A participating ActiveX component has called IObjectContext::DisableCommit.

If a component is not transactional, then SetAbort and SetComplete have the same effect: both cause the component instance to deactivate after the currently executing method returns.

If a method calls none of DisableCommit, EnableCommit, SetAbort, or SetComplete, the default behavior is that of EnableCommit.

See also

DisableCommit, EnableCommit, IsInTransaction, SetAbort

Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide*

IObjectControl interface

Description

Allows components to support EAServer's instance pooling model. The component dispatcher calls the IObjectControl methods to indicate transitions in the lifecycle of an ActiveX component.

Methods

- Activate – Indicates that a component instance has been activated.
- CanBePooled – Determines whether a component instance is eligible for reuse.
- Deactivate – Indicates that a component instance has been deactivated.

Usage

Implement the IObjectControl interface:

- If you want to determine, at runtime, whether a specific instance should be pooled (do not check the Pooling option on the component's Instances tab—otherwise, the CanBePooled method in the IObjectControl interface will not be called), or

- If you need to reset the component's state after deactivation.

Note
To pool instances every time they are deactivated without resetting the component's state, check the Pooling option on the component's Instances tab.

The server can maintain a cache of idle component instances and bind them to individual clients only as needed. This strategy allows the server to service more clients without the performance drain caused by allocating a component instance for each request.

The Activate method indicates that an instance is being removed from the pool to service a client. The Deactivate method indicates that the instance is finished servicing the client. Instance reuse is optional (see "Support for instance pooling" on page 75). However, components that support it will achieve greater scalability.

If you are coding the component in C++, you can directly implement IObjectControl. However, some automation controllers such as PowerBuilder 7.0 provide built-in, implicit support. See your IDE's documentation for more information.

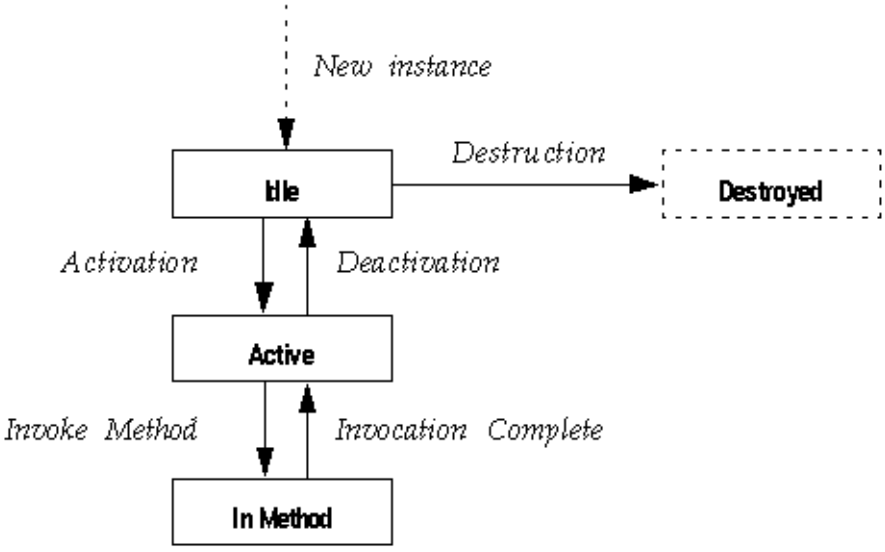
The instance-pooling lifecycle is tightly coupled with the EAServer transaction model. See Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide* for a description of how components participate in transactions.

The next section discusses the ActiveX component lifecycle in detail.

ActiveX component lifecycle

The following figure illustrates the states and state transitions in the lifecycle of an ActiveX component.

Figure 2-1: States in the ActiveX component lifecycle



The state transitions are as follows:

- *New instance* – The EAServer runtime allocates a new instance of the component class. The default constructor is called if one exists. The instance remains idle until the first method invocation.
- *Activation* – Activation prepares a component instance for use by a client. Activate is called. Once an instance is activated, it is bound to one client and can service no other client until it has been deactivated.
- *In Method* – In response to a method invocation request from the client, the EAServer runtime calls the corresponding class method in the component. The next state depends on the method's execution, as follows:
  - If the method throws an uncaught exception, the instance is deactivated. If the method is participating in a transaction, the transaction is rolled back.
  - If the method has called IObjectContext::SetComplete or IObjectContext::SetAbort, the instance is deactivated.

  - If the method has called IObjectContext::EnableCommit or IObjectContext::DisableCommit, the instance is not deactivated. The client's next method invocation is serviced by the same instance unless the client destroys its reference or disconnects.

- *Deactivation* – Deactivation occurs when the instance has called either IObjectContext::SetComplete or IObjectContext::SetAbort, the client has destroyed its stub instance, or the client has disconnected. The EAServer runtime calls the component's Deactivate method to indicate deactivation. After deactivation, the server calls the component's CanBePooled method (unless the Pooling option in the component's Instances tab is checked). If CanBePooled returns TRUE the instance is placed back in the idle pool for reuse. Otherwise, the instance is destroyed.
- *Destruction* – The EAServer runtime destroys the component reference. The component's destructor is called.

Support for instance pooling

To support instance pooling using the IObjectControl interface, you must code your component as follows:

- Code the class to implement the IObjectControl interface.
- Code the CanBePooled method to return `TRUE` if the instance state can be reset.
- In the Activate method, add code to reset any class variables to their initial values, as if the component were freshly constructed. If the component keeps references to stateful objects across activation cycles, you must reset these objects to an initial state as well.

The decision whether to reuse a specific instance can be made at runtime.

Note
CanBePooled is not called if the Pooling option on the component's Instances Tab is checked.

Header file requirements

IObjectControl is defined in *jagctx.h*, which is provided in the EAServer *include* subdirectory.

You must include *initguid.h* in only one source file that is linked into your component DLL. If you do not include *initguid.h* in one file or you include it several files, your project will not link.

initguid.h is not included with EAServer. It is part of the Win32 SDK. Both Microsoft Visual C++ and Powersoft Power++™ provide this file. Other ActiveX C++ builder tools may provide it as well.

See also

IObjectContext interface

Chapter 19, "Creating ActiveX Components," in the *EAServer Programmer's Guide*

IObjectControl::Activate

Description

Indicates that a component instance has been activated.

Syntax

```
#include <jagctx.h>

HRESULT IObjectControl::Activate (void);
```

Return value

Return value	To indicate
S_OK	Success.
Any other value.	Interpreted as an error. If the component is transactional, the component dispatcher rolls back the transaction in which the component is about to participate.

Usage

Activate and Deactivate allow a component's instances to be pooled. If a component supports instance pooling, Activate must reset any class variables to the initial values, as if the component instance were being freshly constructed. To prohibit instance pooling, code the CanBePooled method to return `FALSE`.

See "ActiveX component lifecycle" on page 73 for more information on when Activate and Deactivate are called.

If a component is declared to be transactional and its Activate method returns an error (any value other than S_OK), the component dispatcher rolls back the transaction in which the component is about to participate.

See also

CanBePooled, Deactivate

IObjectControl::CanBePooled

Description

Determines whether a component instance is eligible for reuse.

Note
CanBePooled is not called if the Pooling option on the component's Instances Tab is checked.

Syntax

```
#include <jagctx.h>

BOOL IObjectControl::CanBePooled (void);
```

Return value

Return value	To indicate
TRUE	The instance can be reused.
FALSE	The instance cannot be reused and should be deallocated.

Usage

If a component implements the IObjectControl interface, a single instance can be activated and deactivated many times to serve different clients. After deactivation, the component dispatcher calls the component's CanBePooled method to determine whether the current instance can be reused. If CanBePooled returns FALSE, the dispatcher destroys the instance.

Components that support instance pooling must be coded such that a recycled instance behaves the same as a newly allocated instance. See "Support for instance pooling" on page 75 for more information.

See also

Activate, Deactivate

IObjectControl::Deactivate

Description

Indicates that a component instance has been deactivated.

Syntax

```
#include <jagctx.h>

void IObjectControl::Deactivate (void);
```

Usage

The EAServer runtime calls Deactivate to indicate that the component instance is being deactivated. See "ActiveX component lifecycle" on page 73 for more information on when Activate and Deactivate are called.

If your component caches data changes, you can code the Deactivate method to send cached changes to the remote database server.

Deactivate can be used to deallocate or reset the state of objects that are initialized in the Activate method.

See also

Activate, CanBePooled

ISharedProperty interface

Description — Represents a property value that is shared among all ActiveX component instances in a EAServer package.

Methods
- get_Value – Retrieves a shared property value.
- put_Value – Sets a shared property value.

Usage — Use the ISharedPropertyGroup methods to create or retrieve ISharedProperty objects.

A shared property can be assigned any value that can be represented by an ActiveX VARIANT structure. However, VARIANT values with the VT_BYREF bit set are not allowed.

See also — ISharedPropertyGroup interface, ISharedPropertyGroupManager interface

ISharedProperty::get_Value

Description — Retrieves a shared property value.

Syntax

```
#include <jagctx.h>
#include <JagSharedProp.h>

HRESULT ISharedProperty::get_Value (
    VARIANT* pValue
    );
```

Parameters

pValue
The address of a VARIANT structure to which the property's current value is copied.

Return value

Return value	To indicate
S_OK	Successful retrieval of the property.
E_INVALIDARG	*pValue* was NULL.

See also — put_Value

ISharedProperty::put_Value

Description — Sets a shared property value.

Syntax

```
#include <jagctx.h>
#include <JagSharedProp.h>
```

```
HRESULT ISharedProperty::put_Value (
    VARIANT newValue
    );
```

Parameters

newValue
A VARIANT structure containing the new value for the property.

Return value

Return value	To indicate
S_OK	Successful retrieval of the property.
E_INVALIDARG	The VT_BYREF bit is set in the VARIANT that was passed as *newValue*.
DISP_E_ARRAYISLOCKED	The VARIANT that was passed as *newValue* contains an array that is locked.
DISP_E_BADVARTYPE	The VARIANT that was passed as *newValue* contains an invalid type.

See also

get_Value

ISharedPropertyGroup interface

Description

Represents a group of properties that are shared by all ActiveX components in a EAServer package. Contains methods to create, access, and destroy shared properties.

Methods

- CreateProperty – Creates a new shared property by name.
- CreatePropertyByPosition – Creates a new shared property by position.
- get_Property – Retrieves a reference to a named property.
- get_PropertyByPosition – Retrieves a reference to an indexed property.

Usage

Call the ISharedPropertyGroupManager methods to create a new ISharedPropertyGroup object or to obtain a reference to an existing property group.

Property groups can be shared only among components that are installed in the same EAServer package.

See also

ISharedProperty interface, ISharedPropertyGroupManager interface

ISharedPropertyGroup::CreateProperty

Description

Creates a new shared property by name.

Syntax

```
#include <jagctx.h>
#include <JagSharedProp.h>

HRESULT ISharedPropertyGroup::CreateProperty (
    BSTR propertyName,
    VARIANT_BOOL* pfAlreadyExisted;
    ISharedProperty ** ppProperty,
);
```

Parameters

propertyName

A string containing the name by which the property will be referred.

pfAlreadyExisted

The address of a VARIANT_BOOL variable. On output, set to VARIANT_TRUE if the property already existed or VARIANT_FALSE otherwise. *pfAlreadyExisted* can be NULL if you do not care whether the property existed previously.

ppProperty

On output, a reference to an ISharedProperty object for the property or NULL if an error occurred.

Return value

Return value	To indicate
S_OK	Success.
E_INVALIDARG	Either *name* or *ppProperty* was NULL.

Usage

CreateProperty creates named properties that can be retrieved with the get_Property method. Properties can be referenced either by name or by position but not by both.

Newly created properties are set to a default value, which is a VARIANT of type VT_I4 (4-byte integer), with a value of 0.

Call CreatePropertyByPosition to create indexed properties (retrieved with CreatePropertyByPosition).

See also

CreateProperty, get_Property, CreatePropertyByPosition, ISharedProperty interface

ISharedPropertyGroup::CreatePropertyByPosition

Description

Creates a new shared property by position.

Syntax

```
#include <jagctx.h>
#include <JagSharedProp.h>

HRESULT ISharedPropertyGroup::CreatePropertyByPosition (
    INT position,
    VARIANT_BOOL* pfAlreadyExisted,
    ISharedProperty ** ppProperty
);
```

Parameters

position
The index by which the property will be referred.

pfAlreadyExisted
The address of a VARIANT_BOOL variable. On output, set to VARIANT_TRUE if the property already existed or VARIANT_FALSE otherwise. *pfAlreadyExisted* can be NULL if you do not care whether the property existed previously.

ppProperty
On output, a reference to a ISharedProperty object for the property or NULL if an error occurred.

Return value

Return value	To indicate
S_OK	Success.
E_INVALIDARG	Either *name* or *ppProperty* was NULL.

Usage

CreatePropertyByPosition creates indexed properties that can be retrieved with the get_PropertyByPosition method. Properties can be referenced either by name or by position but not by both means.

Newly created properties are set to a default value, which is a VARIANT of type VT_I4 (4-byte integer), with a value of 0.

Call CreateProperty to create named properties (retrieved with get_Property).

See also

CreateProperty, get_Property, get_PropertyByPosition

ISharedPropertyGroup::get_Property

Description

Retrieves a reference to a named property.

Syntax

```
#include <jagctx.h>
#include <JagSharedProp.h>

HRESULT ISharedPropertyGroup::get_Property (
    BSTR propertyName,
```

```
    ISharedProperty ** ppProp
);
```

Parameters

propertyName
The name of the property to be retrieved.

ppProp
On output, a reference to a ISharedProperty object for the property or NULL if an error occurred.

Return value

Return value	To indicate
S_OK	Success.
E_INVALIDARG	Either *name* or *ppProp* was NULL or no property with the specified name exists in this property group.

Usage

Named properties are created with the CreateProperty method.

get_Property fails if the requested property has not been created. Call CreateProperty when you are not sure whether a property exists yet. CreateProperty retrieves existing properties or creates them if they do not already exist.

See also

CreateProperty, CreatePropertyByPosition, get_PropertyByPosition

ISharedPropertyGroup::get_PropertyByPosition

Description

Retrieves a reference to an indexed property.

Syntax

```
#include <jagctx.h>
#include <JagSharedProp.h>

HRESULT ISharedPropertyGroup::get_PropertyByPosition (
    INT position,
    ISharedProperty ** ppProp
);
```

Parameters

position
The index of the property to be retrieved.

ppProp
On output, a reference to a ISharedProperty object for the property or NULL if an error occurred.

Return value

Return value	To indicate
S_OK	Success.
E_INVALIDARG	*ppProp* was NULL or no property with the specified index exists in this property group.

Usage

Indexed properties are created with the CreatePropertyByPosition method.

get_PropertyByPosition fails if the requested property has not been created. Call CreatePropertyByPosition when you are not sure whether a property exists yet. CreatePropertyByPosition retrieves existing properties or creates them if they do not already exist.

See also

CreateProperty, CreatePropertyByPosition, get_Property

ISharedPropertyGroupManager interface

Description

Contains methods to create, access, and destroy shared property groups.

Methods

- CreatePropertyGroup – Creates a new property group or retrieve a reference to the existing group with the specified name.
- get_Group – Retrieves a reference to an existing property group.
- get__NewEnum – Not supported.

Usage

The ISharedPropertyGroupManager interface allows you to create new shared property groups and find out about existing groups.

To create a ISharedPropertyGroupManager interface pointer, use the ProgID, "Jaguar.SharedPropertyGroupManager." Call the OLE routines CLSIDfromProgID and CoCreateInstance. CoCreateInstance returns an interface pointer for a given ActiveX class ID string. CLSIDfromProgID obtains the class ID string that CoCreateInstance requires.

To use the ISharedPropertyGroupManager, ISharedPropertyGroup, and ISharedProperty interfaces, you must include *JagSharedProp.h*. Additionally, you must include *JagSharedProp_i.c* in only one source file for your component DLL. *JagSharedProp.h* contains interface definitions for the documented interfaces. *JagSharedProp_i.c* declares symbols that are required by the ActiveX CoCreateInstance routine.

Warning! You will get duplicate-symbol link errors if you include *JagSharedProp_i.c* in more than one source file for your component DLL.

See also ISharedProperty interface, ISharedPropertyGroup interface

ISharedPropertyGroupManager::CreatePropertyGroup

Description Creates a new property group or retrieve a reference to the existing group with the specified name.

Syntax

```
#include <jagctx.h>
#include <JagSharedProp.h>

HRESULT ISharedPropertyGroupManager::CreatePropertyGroup (
        BSTR groupName,
        LONG* plIsolationMode,
        LONG* plReleaseMode,
        VARIANT_BOOL* pfExists,
        ISharedPropertyGroup ** ppGroup
);
```

Parameters

groupName

The name by which the property group is referred. Cannot be NULL, but a zero-length string is a valid name.

plIsolationMode

A pointer to a LONG variable that describes the isolation (locking) mode for access to the property group. On input, must be the following symbolic constant:

Isolation mode	Value	Meaning
LockMethod	1	The property group is locked from the first access until the current method returns. Use this isolation mode to prevent other component instances from accessing a property group while you retrieve or set multiple properties in the group.

If the property group already exists, the input value is ignored and output value is set to reflect the isolation mode of the existing property group.

plReleaseMode

A pointer to a LONG variable that describes the release mode for the property group. On input, must be the following symbolic constant:

Release mode	Value	Meaning
`Process`	1	The property group is not destroyed even when all references have been released.

If the property group already exists, the input value is ignored and output value is set to reflect the release mode of the existing property group.

pfAlreadyExisted

On output, set to VARIANT_TRUE if the property group already existed or VARIANT_FALSE otherwise. Can be NULL if you do not care whether the group existed previously.

ppGroup

The address of a ISharedPropertyGroup interface pointer. On output, contains a reference to a ISharedPropertyGroup object for the property group or NULL if an error occurred.

Return value

Return value	To indicate
S_OK	Success.
E_INVALIDARG	One or more parameters contained invalid input values.

Usage

CreatePropertyGroup creates a new shared property group or returns a reference to an existing group that has the specified name.

Property groups can be shared only among components that are installed in the same EAServer package. A group created by a component that is installed in one package cannot be retrieved by a component that is installed in a different package.

See also

get_Group, ISharedPropertyGroup interface

ISharedPropertyGroupManager::get_Group

Description

Retrieves a reference to an existing property group.

Syntax

```
#include <jagctx.h>
#include <JagSharedProp.h>
```

```
HRESULT ISharedPropertyGroupManager::get_Group (
    BSTR name,
    ISharedPropertyGroup ** ppGroup,
);
```

Parameters

name

The name of the group to be retrieved.

ppGroup

On output, a reference to a ISharedPropertyGroup object for the property group or NULL if an error occurred.

Return value

Return value	To indicate
S_OK	Success.
E_INVALIDARG	Either *name* or *ppGroup* was NULL or no property group exists with the specified name.

Usage

get_Group returns a reference to the property group with the same name. get_Group fails if no group has been created with the specified name. Call CreatePropertyGroup when you are not sure whether a group already exists.

Property groups can be shared only among components that are installed in the same EAServer package. A group created by a component that is installed in one package cannot be retrieved by a component that is installed in a different package.

See also

CreatePropertyGroup, ISharedPropertyGroup interface

CHAPTER 3

ActiveX IDispatch Interface Reference

This chapter documents the IDispatch interfaces for EAServer's server-side ActiveX objects. The IDispatch interface is used by ActiveX automation controllers such as Microsoft Visual Basic.

Most objects also provide a custom interface defined in a C++ header file. See Chapter 2, "ActiveX C++ Interface Reference" for descriptions of these interfaces.

How to use these reference pages

These reference pages show the syntax of method calls using Microsoft's Visual Basic language. For other development tools, use the tool's OLE object browser to see method syntax displayed as appropriate for the tool's script syntax.

The reference page for each interface will list the interface's ProgID and the name of the type library that defines it. You may need this information to create object references. For example, in Visual Basic, you must add references to the project for each EAServer type library that contains an interface used by your application. In your Visual Basic code, objects that implement the interface can be declared using this syntax:

```
Dim myobject As typelib.interface
```

where *typelib* is the name of the type library that defines the interface, and *interface* is the name of the interface. If code that follows this rule does not compile, you most likely have not added a reference to the type library in your project.

IDispatch interface index

- IJagServer interface – Contains utility methods for use in EAServer ActiveX components.
- IJagServerResults interface – Provides methods to send rows to a EAServer client application.
- SharedProperty interface – Represents a property value that is shared among all ActiveX component instances in a EAServer package.
- SharedPropertyGroup interface – Represents a group of properties that are shared by all ActiveX components in a EAServer package. Contains methods to create, access, and destroy shared properties.
- SharedPropertyGroupManager interface – Contains methods to create, access, and destroy shared property groups.

IJagServer interface

Description

Type Library	JAGAXWrapLib
ProgID	Jaguar.JagServer

Contains utility methods for use in EAServer ActiveX components.

Methods
- WriteLog – Writes a message to the server's log file.

See also
Chapter 19, "Creating ActiveX Components," in the *EAServer Programmer's Guide*

IJagServer.WriteLog

Description
Writes a message to the server's log file.

Syntax
IJagServer.WriteLog(useTimeStamp, message)

Parameters

useTimeStamp
`TRUE` if the current date and time should be prepended to the log message; `FALSE` otherwise.

message
A message to be written to the server's log file.

Examples

The following Visual Basic fragment declares a function that writes a string to the server's log, prepended with the name of the component:

```
Private Function writeToLog(msg As String)
  Dim jserver As JAGAXWrapLib.JagServer
  Set jserver = New JAGAXWrapLib.JagServer
  jserver.WriteLog True, Format("MyComponent: " & msg)
End Function
```

See also

Chapter 19, "Creating ActiveX Components," in the *EAServer Programmer's Guide*

IJagServerResults interface

Description

Type Library	JAGAXWrapLib
ProgID	Jaguar.JagServerResults

Provides methods to send rows to a EAServer client application.

Methods

- BeginResults – Begins the sequence of calls that sends a result set to the client.
- BindCol – Deprecated equivalent of BindColumn. The two methods are equivalent, except that the BindCol *sourceBuf* and *indicator* parameters are not explicitly declared [in, out] in the type library.
- BindColumn – Binds a program variable to a column in a result set.
- ColAttributes – Specifies additional metadata for a column to be sent in a result set.
- DescribeCol – Describes a result set column.
- EndResults – Indicates that all rows in a result set have been sent.
- ResultsPassthru – Deprecated equivalent of ResultSetsPassthrough. The methods are equivalent except that the ResultsPassthrough *pInfo* parameter is not explicitly declared [in, out] in the type library.
- ResultSetsPassthrough – Forwards results from a remote database query to the client.
- SendData – Sends one row in a result set.

See also

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServerResults.BeginResults

Description
Begins the sequence of calls that sends a result set to the client.

Syntax
IJagServerResults.BeginResults(ByVal numColumns as Integer)

Parameters
numColumns
An integer that specifies the number of columns in the result set to be sent.

See also
EndResults

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServerResults.BindCol

Description
Deprecated equivalent of BindColumn. The two methods are equivalent, except that the BindCol *sourceBuf* and *indicator* parameters are not explicitly declared as [in, out] in the type library.

Syntax
```
IJagServerResults.BindColumn(
    ByVal itemNumber As Integer,
    sourceBuf,
    ByVal maxBuflen As Integer,
    indicator As Integer
)
```

Parameters
itemNumber
See BindColumn.

sourceBuf
See BindColumn.

maxBufLen
See BindColumn.

indicator
See BindColumn.

Usage
BindCol is a deprecated equivalent of BindColumn. The two methods are equivalent, except that the BindCol *sourceBuf* and *indicator* parameters are not explicitly declared as [in, out] in the type library. You must use BindColumn in Visual Basic applications and with most other tools that use the ActiveX IDispatch interface.

See also
BindColumn

IJagServerResults.BindColumn

Description

Binds a program variable to a column in a result set.

Syntax

```
IJagServerResults.BindColumn(
    ByVal itemNumber As Integer,
    sourceBuf,
    ByVal maxBuflen As Integer,
    indicator As Integer
)
```

Parameters

itemNumber

An integer specifying the column number. The first column is 1.

sourceBuf

A variable to supply values for this column when row data is sent with the SendData method. The variable must be of a datatype that can be converted to the SQL datatype that was specified when DescribeCol was called for the column. “ActiveX to SQL datatype conversions” on page 92 lists SQL datatypes and allowable bind types. You must use a different variable for each column, even if two columns have the same datatype.

maxBufLen

For character and binary data, the maximum length that values for this column can have.

indicator

An integer passed by reference. Before calling the SendData method to send each row, set the *indicator* variable to indicate whether the column’s current value is null. You must use a different indicator variable for each column. Indicator values are as follows:

Value	To indicate
0	Column value is not null and must be read from the *sourceBuf* `Variant` buffer.
-1	Column value is null.

Usage

BindCol associates a program variable with a column in a result set. When SendData is called to send a row, it reads the column value for the current row from the variable that is bound to the column.

SQL Datatype / Automation Datatype	BSTR	VARIANT_BOOL	short	long	float	double	SAFEARRAY	DATE
"SQL_CHAR"	X	X	X	X	X	X	X	X
"SQL_VARCHAR"	X	X	X	X	X	X	X	X
"SQL_LONGVARCHAR"	X	X	X	X	X	X	X	X
"SQL_DECIMAL"	X	X	X	X	X	X	X	
"SQL_NUMERIC"	X	X	X	X	X	X	X	
"SQL_BIT"	X	X	X	X	X	X	X	
"SQL_TINYINT"	X	X	X	X	X	X	X	
"SQL_SMALLINT"	X	X	X	X	X	X	X	
"SQL_INTEGER"	X	X	X	X	X	X	X	
"SQL_REAL"	X	X	X	X	X	X	X	
"SQL_FLOAT"	X	X	X	X	X	X	X	
"SQL_DOUBLE"	X	X	X	X	X	X	X	
"SQL_BINARY"	X	X	X	X	X	X	X	
"SQL_VARBINARY"	X	X	X	X	X	X	X	
"SQL_LONGVARBINARY"	X	X	X	X	X	X	X	
"SQL_DATE"	X						X	X
"SQL_TIME"	X						X	
"SQL_TIMESTAMP"	X						X	X

ActiveX to SQL datatype conversions

The *SQLDatatype* value passed to DescribeCol determines the datatype with which column values are sent over the network. If the program variable type does not map directly to the column's SQL datatype, SendData attempts to convert the value. The figure below shows the supported conversions between SQL datatypes and bind variable types. An X indicates a supported conversion.

See also

ColAttributes, DescribeCol, SendData

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServer.ColAttributes

Description

Specifies additional metadata for a column to be sent in a result set.

Syntax

```
IJagServerResults.ColAttributes(
    ByVal itemNumnber as Integer,
    “COLUMN_MONEY”,
    ByVal trueFalse as Boolean
)
```

Parameters

itemNumber

An integer specifying the column number. The first column is 1.

trueFalse

A Boolean value. A value of True means that the column represents a cash value.

See also

DescribeCol

Chapter 25, “Sending Result Sets,” in the *EAServer Programmer’s Guide*

IJagServerResults.DescribeCol

Description

Describes a result set column.

Syntax

```
IJagServerResults.DescribeCol(
    ByVal itemNumber as Integer,
    ByVal columnName as String,
    ByVal SQLDatatype as String,
    ByVal columnSize as Long,
    ByVal precision as Long,
    ByVal scale as Long,
    ByVal nullable as Boolean
)
```

Parameters

itemNumber

An integer specifying the column number. The first column is 1.

columnName

A string specifying the column’s name.

SQLDatatype

A string specifying the name of the column’s SQL datatype. This value determines the datatype of values sent to the client. Values are specified in a buffer that is bound to the column with BindColumn. The table below lists the SQL datatype strings. See BindColumn for details on how to bind column values.

Table 3-1: SQL datatypes for DescribeCol

SQL datatype string	Description
“SQL_BIT”	A boolean value (1 bit of data).
“SQL_TINYINT”	A one-byte integer.
“SQL_SMALLINT”	A two-byte integer.
“SQL_INTEGER”	A four-byte integer.
“SQL_REAL”	A four-byte IEEE floating point value.
“SQL_FLOAT”	An 8-byte IEEE floating point value.
“SQL_DOUBLE”	Same as “SQL_FLOAT”
“SQL_NUMERIC”	A fixed-point fractional number with precision and scale specified by parameters *precision* and *scale*.
“SQL_DECIMAL”	Same as “SQL_NUMERIC”.
“SQL_CHAR”	A string of a fixed length not greater than 255 characters. Values shorter than the length specified by *columnSize* are padded with spaces.
“SQL_VARCHAR”	A string of varying length, limited to 255 characters in length.
“SQL_LONGVARCHAR”	A string of varying length, with no length limit.
“SQL_DATE”	An ODBC date value.
“SQL_TIME”	An ODBC time value.
“SQL_TIMESTAMP”	An ODBC timestamp value.
“SQL_BINARY”	An array of bytes, whose values have a fixed length not greater than 255 characters.
“SQL_VARBINARY”	An array of bytes, whose values have varying length not greater than 255 characters.
“SQL_LONGVARBINARY”	An array of bytes, whose values have varying length with no length limit.

columnSize

For character or binary columns, a long integer that specifies the maximum length for column values.

precision

A long integer that specifies the precision of column values. For “SQL_NUMERIC” or “SQL_DECIMAL” columns, *precision* indicates the maximum number of decimal digits that a value may have. For other datatypes, *precision* is ignored.

scale
An integer that specifies the scale for column values. For "SQL_NUMERIC" or "SQL_DECIMAL" columns, *scale* indicates the number of decimal digits to the right of the decimal point. For other datatypes, *scale* is ignored.

nullable
A Boolean value that specifies whether the column can contain null values. True indicates that the column may have null values.

See also BindColumn, ColAttributes

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServer.EndResults

Description Indicates that all rows in a result set have been sent.

Syntax
```
IJagServerResults.EndResults(ByVal rowCount as Long)
```

Parameters *rowCount*
A positive long integer that specifies the number of rows that were sent to the client.

See also BeginResults, SendData

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServer.ResultsPassthru

Description Deprecated equivalent of ResultSetsPassthrough. The methods are equivalent except that the ResultsPassthrough *pInfo* parameter is not explicitly declared [in, out] in the type library.

Syntax
```
IJagServerResults.ResultsPassthrough(
    ByVal conlibName as String,
    ByVal conlibPtr,
    ByVal pthruType as String,
    pInfo as Long
)
```

Parameters *conlibName*
See ResultSetsPassthrough.

conlibPtr
See ResultSetsPassthrough.

pthruType
See ResultSetsPassthrough.

pInfo
See ResultSetsPassthrough.

Usage

ResultsPassthrough is a deprecated equivalent of ResultSetsPassthrough. The methods are equivalent except that the ResultsPassthrough *pInfo* parameter is not explicitly declared [in, out] in the type library.

You must use ResultSetsPassthrough in Visual Basic and other tools that use the IDispatch interface.

See also

ResultSetsPassthrough

IJagServer.ResultSetsPassthrough

Description

Forwards results from a remote database query to the client.

Syntax

```
IJagServerResults.ResultsPassthrough(
    ByVal conlibName as String,
    ByVal conlibPtr,
    ByVal pthruType as String,
    pInfo as Long
)
```

Parameters

conlibName
A string with one of the following values:

- "ODBC" to indicate that *conlibPtr* contains the address of an ODBC HSTMT control structure.
- "CTLIB" to indicate that *conlibPtr* contains the address of a Client-Library CS_COMMAND control structure.

conlibPtr
The "handle" for the connectivity library control structure used to retrieve results. When using ODBC, pass a handle to an HSTMT control structure as *conlibPtr*. The HSTMT must be in a state that allows SQLFetch to be called without error.

When using Client-Library, set *conlibPtr* to a Client-Library CS_COMMAND control structure. The CS_COMMAND must be in a state that allows ct_results to be called without error.

pthruType

A string with one of the following values:

- "CURRENT_RESULTS" to indicate that only the current result set should be forwarded to the client. When using this option, you must ensure that all result sets are processed. You can call ResultSetsPassthrough in a loop (see examples in "Comments" below). You can also retrieve or cancel subsequent result sets by directly calling ODBC or Client-Library routines.
- "ALL_RESULTS" to indicate that all result sets should be forwarded to the client.

pInfo

Pass as NULL if using ODBC or whenever using the "ALL_RESULTS" option to forward all results with one call.

When forwarding individual Client-Library result sets, you must pass an integer variable by reference as *pInfo*. ResultSetsPassthrough sets the *pInfo* variable to specify whether all results have been retrieved from the CS_COMMAND structure, as follows:

pInfo value	To indicate
`NoInfo` (0)	More results remain to be processed.
`NoMoreResults` (1)	All results have been processed.

Usage

ResultSetsPassthrough forwards ODBC or Client-Library result sets to the client.

All results from a query can be forwarded with one call using the "ALL_RESULTS" option for the *pthruType* parameter. To forward single result sets, use the "CURRENT_RESULTS" option.

See also

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

IJagServerResults.SendData

Description

Sends one row in a result set.

Syntax

```
IJagServerResults.SendData();
```

Usage

SendData sends a row of data to the client. Values for each are read from the variables that were bound to the column with the BindColumn method.

See also

BeginResults, DescribeCol, EndResults

Chapter 25, “Sending Result Sets,” in the *EAServer Programmer’s Guide*

SharedProperty interface

Description

Type Library	JAGSHAREDPROPLib
ProgID	Jaguar.SharedProperty

Represents a property value that is shared among all ActiveX component instances in a EAServer package.

Properties

- Value – The value of the property. A shared property can be assigned any value that can be represented by an ActiveX VARIANT structure. However, VARIANT values with the VT_BYREF bit set are not allowed.

Usage

Call the SharedPropertyGroup interface methods to create a new SharedProperty object or to obtain a reference to an property.

See also

SharedPropertyGroup interface, SharedPropertyGroupManager interface

SharedPropertyGroup interface

Description

Type Library	JAGSHAREDPROPLib
ProgID	Jaguar.SharedPropertyGroup

Represents a group of properties that are shared by all ActiveX components in a EAServer package. Contains methods to create, access, and destroy shared properties.

Methods

- CreateProperty – Creates a new shared property by name.
- CreatePropertyByPosition – Creates a new shared property by position.
- Property – Retrieves a reference to a named property.
- PropertyByPosition – Retrieves a reference to an indexed property.

Usage

Call the SharedPropertyGroupManager methods to create a new SharedPropertyGroup object or to obtain a reference to an existing property group.

Property groups can be shared only among components that are installed in the same EAServer package.

See also SharedProperty interface, SharedPropertyGroupManager interface

SharedPropertyGroup.CreateProperty

Description Creates a new shared property by name.

Syntax

```
Dim myProp as JAGSHAREDPROPLib.SharedProperty

myProp = SharedPropertyGroup.CreateProperty (
    ByVal propertyName as String,
    alreadyExisted as Boolean)
```

Parameters

myProp

A variable to receive the SharedProperty interface pointer. On return, the variable is a SharedProperty that accesses the property, or NULL if an error occurred.

propertyName

Specifies the name by which the property is referred.

alreadyExisted

A Boolean variable passed by reference. On output, set to TRUE if the group existed before the call, and FALSE otherwise.

Usage CreateProperty creates named properties that can be retrieved with the Property method. Properties can be referenced either by name or by position but not by both.

Newly created properties are set to a default value, which is a VARIANT of type VT_I4 (4-byte integer), with a value of 0.

Call CreatePropertyByPosition to create indexed properties (retrieved with PropertyByPosition).

See also CreatePropertyByPosition, Property, PropertyByPosition, SharedProperty interface

SharedPropertyGroup.CreatePropertyByPosition

Description Creates a new shared property by position.

Syntax

```
Dim myProp as JAGSHAREDPROPLib.SharedProperty
```

```
myProp = SharedPropertyGroup.CreatePropertyByPosition(
    ByVal position as Integer,
    alreadyExisted as Boolean)
```

Parameters

myProp
A variable to receive the SharedProperty interface pointer. On return, the variable is a SharedProperty that accesses the property, or NULL if an error occurred.

position
The integer index by which the property is referred.

alreadyExisted
A Boolean variable passed by reference. On output, set to TRUE if the group existed before the call, and FALSE otherwise.

Usage

CreatePropertyByPosition creates indexed properties that can be retrieved with the PropertyByPosition method. Properties can be referenced either by name or by position but not by both.

Newly created properties are set to a default value, which is a VARIANT of type VT_I4 (4-byte integer), with a value of 0.

Call CreateProperty to create named properties (retrieved with Property).

See also

CreateProperty, Property, PropertyByPosition

SharedPropertyGroup.Property

Description

Retrieves a reference to a named property.

Syntax

```
Dim myProp as JAGSHAREDPROPLib.SharedProperty

myProp = SharedPropertyGroup.CreateProperty(
    ByVal name as String)
```

Parameters

myProp
A variable to receive the SharedProperty interface pointer. On return, the variable is set to a SharedProperty that accesses the property, or NULL if an error occurred.

name
The name of the property to be retrieved.

Usage

Named properties are created with the CreateProperty method.

Property fails if the requested property has not been created. Call CreateProperty when you are not sure whether a property exists yet. CreateProperty retrieves existing properties or creates them if they do not already exist.

See also CreateProperty, CreatePropertyByPosition, PropertyByPosition

SharedPropertyGroup.PropertyByPosition

Description Retrieves a reference to an indexed property.

Syntax
```
Dim myProp as JAGSHAREDPROPLib.SharedProperty

myProp = SharedPropertyGroup.PropertyByPosition(
    ByVal position as Integer)
```

Parameters

myProp
A variable to receive the SharedProperty interface pointer. Set to a SharedProperty that accesses the property or NULL if an error occurred.

position
The index of the property to be retrieved.

Usage Indexed properties are created with the CreatePropertyByPosition method.

PropertyByPosition fails if the requested property has not been created. Call CreatePropertyByPosition when you are not sure whether a property exists yet. CreatePropertyByPosition retrieves existing properties or creates them if they do not already exist.

See also CreateProperty, CreatePropertyByPosition, Property

SharedPropertyGroupManager interface

Description

Type Library	JAGSHAREDPROPLib
ProgID	Jaguar.SharedPropertyGroupManager

Contains methods to create, access, and destroy shared property groups.

Methods

- CreatePropertyGroup – Creates a new property group or retrieve a reference to the existing group with the specified name.
- Group – Retrieves a reference to an existing property group.

Usage — The SharedPropertyGroupManager interface allows you to create new shared property groups and find out about existing groups.

See also — SharedProperty interface, SharedPropertyGroup interface

SharedPropertyGroupManager.CreatePropertyGroup

Description — Creates a new property group or retrieve a reference to the existing group with the specified name.

Syntax

```
Dim propgroup as JAGSHAREDPROPLib.SharedPropertyGroup

propgroup = SharedPropertyGroupManager.CreatePropertyGroup(
    ByVal groupName as String,
    isololationMode as Integer,
    releaseMode as Integer,
    alreadyExisted as Boolean)
```

Parameters

propGroup

A variable to receive the SharedPropertyGroup interface pointer for the new or existing property group. Set to NULL if an error occurs.

groupName

A string initialized to the name by which the property group is referred. A zero-length string is a valid name.

isolationMode

An `Integer` variable passed by reference. Specifies the isolation mode, which determines how properties within the group are accessed. The input value must be the following symbolic constant:

Isolation mode	Value	Meaning
`LockMethod`	1	The property group is locked from the first access until the current method returns. Use this isolation mode to prevent other component instances from accessing a property group while you retrieve or set multiple properties in the group.

If the property group already exists, the input value is ignored and the output value is set to reflect the isolation mode of the existing property group.

releaseMode

An `Integer` variable passed by reference. The variable describes the release mode for the property group. On input, must be the following symbolic constant:

Release mode	Value	Meaning
`Process`	1	The property group is not destroyed even when all references have been released.

If the property group already exists, the input value is ignored and output value is set to reflect the release mode of the existing property group.

alreadyExisted

A `Boolean` variable passed by reference. On output, set to TRUE if the property group already existed or FALSE otherwise.

Usage

CreatePropertyGroup creates a new shared property group or returns a reference to an existing group that has the specified name.

Property groups can be shared only among components that are installed in the same EAServer package. A group created by a component that is installed in one package can not be retrieved by a component that is installed in a different package.

See also

Group, SharedPropertyGroup interface

SharedPropertyGroupManager.Group

Description

Retrieves a reference to an existing property group.

Syntax

```
Dim propgroup as JAGSHAREDPROPLib.SharedPropertyGroup

propgroup = SharedPropertyGroupManager.Group(
    ByVal groupName as String)
```

Parameters

propGroup

A variable to receive the SharedPropertyGroup interface pointer for the property group. Set to NULL if an error occurs.

groupName

The name of the group to be retrieved. A zero-length string is a valid name.

Usage

Group returns a reference to the property group with the same name. Group fails if no group has been created with the specified name. Call CreatePropertyGroup when you are not sure whether a group already exists.

Property groups can be shared only among components that are installed in the same EAServer package. A group created by a component that is installed in one package can not be retrieved by a component that is installed in a different package.

See also

CreatePropertyGroup, SharedPropertyGroup interface

CHAPTER 4

ActiveX Client Interfaces

This chapter documents the interfaces that ActiveX clients use to interact with EAServer components.

For an overview of ActiveX clients, see Chapter 20, "Creating ActiveX Clients," in the *EAServer Programmer's Guide*.

How to use these reference pages

These reference pages show the syntax of method calls in the Microsoft Visual Basic language. For other development tools, use the tool's OLE object browser to see method syntax displayed as appropriate for the tool's script syntax. The reference page for each interface lists the ActiveX type library that defines the interface and the interface's ProgID.

The reference page for each interface lists the interface's ProgID and the name of the type library that defines it. You may need this information to create object references. For example, in Visual Basic, you must add references to the project for each EAServer type library that contains an interface used by your application. In your Visual Basic code, objects that implement the interface can be declared using this syntax:

```
Dim myobject As typelib.interface
```

where *typelib* is the name of the type library that defines the interface, and *interface* is the name of the interface. If code that follows this rule does not compile, you most likely have not added a reference to the type library in your project.

Interface index

- Field – Represents one column in a row of tabular data. Modeled after Field in Microsoft's ActiveX Data Objects (ADO) interface.

- Fields – A collection of Field objects that represents a row of tabular data. Modeled after Fields in Microsoft's ActiveX Data Objects (ADO) interface.
- JagORBClientErrNum – Defines symbolic constants for errors that can occur in the ActiveX client proxy.
- JagORBSrvErrNum – Defines symbolic constants for errors that can occur during server-side execution of a method call.
- JCollection – Represents a collection of objects or primitive data values; the ActiveX mapping for CORBA IDL sequences used as method parameter or return types in EAServer component methods.
- Object – A generic proxy object that must be narrowed to another interface.
- Orb – The core interface used by clients that use CORBA-style proxy instantiation.
- RecordSet – Represents a set of tabular data returned by a component method invocation. Provides methods to iterate through the rows in each result set. Modeled after RecordSet in Microsoft's ActiveX Data Objects (ADO) interface.

Field interface

Description

Type library name	*JaguarTypeLibrary*
DLL name	*jagproxy.dll*

Represents one column in a row of tabular data.

Properties

- Type – Integer. Returns a constant that indicates the column's datatype. Table 4-1 lists possible values.
- Value – Variant. Returns the column's value in the Variant type that matches the column's database type. Table 4-1 lists SQL datatypes and the corresponding Variant types.
- ActualSize – Integer. For string and binary data, returns the length of the current value.
- DefinedSize – Integer. For string and binary data, returns the maximum length that values in the column may have.

- Name – String. Returns the column's name.
- NumericScale – Integer. For fixed-precision numeric values, returns the column's scale. The scale is the number of decimal digits to the right of the decimal point.
- OriginalValue – Same as the Value property.
- Precision – Integer. For fixed-precision numeric values, returns the column's precision. The precision is the number of decimal digits in the value.
- UnderlyingValue – Same as the Value property.

Usage

For sample code that accesses a Field object's properties, see Chapter 20, "Creating ActiveX Clients," in the *EAServer Programmer's Guide*.

Table 4-1 lists SQL datatypes, the corresponding values for the Type property, and the corresponding Variant datatypes for the Value property. Values for the type property are defined in the DataTypeEnum enumeration. The table lists both the symbolic DataTypeEnum values and the numeric constants that they represent. Some automation controllers may not be able to use symbolic values from an enumeration; in these controllers, use the numeric constant instead.

Table 4-1: The Field.Type and Field.Value properties

SQL datatype	Field.Type constant	Field.Value return type
BIT, or 1 bit of data	adBoolean (11)	VT_BOOL (1 maps to `true`)
TINYINT, an 1-byte integer	adTinyInt (16)	VT_UI1
SMALLINT, a 2-byte integer	adSmallInt (2)	VT_I2
INTEGER, a 4-byte integer	adInteger (3)	VT_I4
FLOAT, an 8-byte floating point number	adDouble (5)	VT_R8
CHAR, string values that do not vary in length	adChar (129)	VT_BSTR
VARCHAR, string values that can vary in length	adVarChar (200)	VT_BSTR
BINARY, an array of bytes that does not vary in length	adBinary (128)	VT_ARRAY

SQL datatype	Field.Type constant	Field.Value return type
VARBINARY, an array of bytes that may vary in length	adVarBinary (204)	VT_ARRAY
NUMERIC, a fixed-point decimal number	adNumeric (131)	VT_R8 (No direct mapping exists. Mapped to 8-byte floating point)
DECIMAL, a fixed-point decimal number	adDecimal (14)	VT_R8 (No direct mapping exists. Mapped to 8-byte floating point)
DATE, a date value including the time-of-day	adDate (7)	VT_DATE
MONEY, a cash value	adCurrency (6)	VT_CY

See also RecordSet interface, Fields

Fields collection

Description

Type library name	*JaguarTypeLibrary*
DLL name	*jagproxy.dll*

A collection of Field objects that represents a row of tabular data. Modeled after Fields in Microsoft's ActiveX Data Objects (ADO) interface.

Properties

- Count – Integer. Returns the number of columns in the row.
- Item – Returns the Field object that represents the column at a given position within the row.

Usage

The RecordSet Fields property returns Fields collections.

For example code, see Chapter 20, "Creating ActiveX Clients," in the *EAServer Programmer's Guide*.

See also RecordSet interface, Field

Fields.Item

Description — Returns the Field object that represents the column at a given position within the row. Modeled after Field in Microsoft's ActiveX Data Objects (ADO) interface.

Syntax — Fields.Item(index)

Parameters — *index*

An integer that specifies the position of the column of interest. The first column in the row is 0. The last item is Field.Count - 1.

See also — Field

JagORBClientErrNum enumeration

Description

Type library name	*JaguarTypeLibrary*
DLL name	*jagproxy.dll*

Defines symbolic constants to errors that can occur in the ActiveX client proxy.

Table 4-2 lists the codes for client-side error numbers defined in the JagORBClientErrNum enumeration:

Table 4-2: JagORBClientErrNum error codes

Symbolic error code	Number	Description
jagClNonByteArrayErr	8000	Method arguments of type array can only have a base element type of byte.
jagClMultiDimArrayErr	8001	Multidimensional arrays not supported as an argument to a method.
jagClArrayRedimErr	8002	A fatal internal error was encountered while attempting to resize a method argument of type array.
jagClArrayProcErr	8003	A fatal internal error was encountered while processing a method argument of type array.
jagClArrayEmptyErr	8004	An array of size 0 was passed as parameter to a method.
jagClArrayBoundsErr	8005	A fatal internal error was encountered while attempting to determine the upper bound on a method argument of type array.

Symbolic error code	Number	Description
jagClNotJagComponentErr	8006	The component being instantiated is not a valid EAServer component or was not registered in the Windows Registry.
jagClOutOfMem	8007	The application failed to acquire memory from the operating system.
jagClCreateFactErr	8008	The EAServer proxy server cannot instantiate a Factory object. Please contact Sybase Technical Support.
jagClTypeLibErr	8009	The type library for the Component cannot be read from the NT Registry. Please check if a valid directory location was specified for the Type Library while registering the component.
jagClTypeInfoErr	8010	The type information for the Component cannot be read from the Type Library. Regenerate TLB and REG files for the component.
jagClMethInfoErr	8011	The metadata for the method or component cannot be read from the NT Registry or the method is using parameter types that are not presently supported in the EAServer ActiveX proxy.
jagClMethNameErr	8012	The metadata for the method invoked on component cannot be read from the NT Registry. Regenerate TLB and REG files for the component.
jagClCompNameErr	8013	The component name for the component being instantiated cannot be read from the NT Registry.
jagClPkgNameErr	8014	The package name for the Component being instantiated cannot be read from the NT Registry.
jagClPxyCreateErr	8015	Component creation failed.
jagClPxyDestroyErr	8016	Component deletion failed.
jagClPxyFuncDescErr	8017	The metadata information for the method cannot be read from the type library.
jagClArgCountErr	8018	There was a mismatch between the number of parameters passed to method and the number of parameters as described by the information in the type library.

Symbolic error code	Number	Description
jagClInternalErr	8019	An error was encountered while invoking a component method.
jagClParamInfoErr	8020	The type information for a method parameter cannot be read from the Type Library.
jagClTypeMismatchErr	8021	There is a mismatch between type of the value passed as an argument with its specified type in the Type Library.
jagClConversionErr	8022	The data conversion attempted is presently not supported.
jagClArgUpdateErr	8023	An error was encountered while updating an input-output or output parameter for a method.
jagClRetValSetErr	8024	An error was encountered while updating the return value for a method.
jagClRecsetArgErr	8025	The ResultSet type cannot be passed as a parameter in either the input or input-output modes by a EAServer ActiveX application.
jagClUnsuppTypeErr	8026	An unsupported OLE Automation type was used as a parameter in a method.
jagClAxConvertErr	8027	An error was encountered while converting a input-output method parameter received from the server.
jagClJagConvertErr	8028	An error was encountered while converting a input parameter prior to method invocation.
jagClNoInitErr	8029	A component instance must be created prior to invoking a method.
jagClRecordsetCreateErr	8030	An internal error was encountered while creating the Recordset object.
jagClRecordsetMoveErr	8031	Attempt to call MoveNext on a RecordSet which has its EOF property as TRUE.
jagClIteratorPosErr	8032	An invalid position was specified while attempting to retrieve an element from a collection.
jagClInvalidMethodErr	8033	The only method supported on the generic EAServer Object type is Narrow_.

Symbolic error code	Number	Description
jagClNarrowFailErr	8034	The object reference cannot be narrowed to the interface name specified.
jagClInvalidIntfErr	8035	The fully scoped interface name passed as an argument to the Narrow_ method is invalid.
jagClOrbInitErr	8036	An internal error was encountered while initializing client-side ORB.
jagClOrbStrToObjErr	8037	An internal error was encountered while invoking the ORB.string_to_object method.
jagClNotJagCollErr	8038	he parameter of the sequence type passed to the method is not a valid EAServer ActiveX Collection Object.
jagClInternalCollErr	8039	A fatal internal error occured while performing an operation on a EAServer Collection object.
jagClAxSSLCBRegErr	8040	A fatal internal error occured while registering user's ActiveX SSL Callback component. Verify that the directory containing the file *jagproxy.dll* is in the PATH.
jagClDuplicAxSSLCBCompErr	8041	The "AXSSLCBComponent" ORB property cannot be set more than once per session. An ActiveX SSL Callback component with a ProgID of *progid* has previously been registered for the present client session.

Usage

In Visual Basic, exceptions are mapped to the built-in Err object. The exception number maps to Err.Number and the description is available as Err.Description. You can handle exceptions by activating error handling code with On Error Goto statement or by checking whether Err.Number is > 0.

The proxy type library defines error numbers for client-side errors in the JagORBClientErrNum enumeration and server-side error numbers in the JagORBSrvErrNum enumeration.

See also

JagORBSrvErrNum enumeration

JagORBSrvErrNum enumeration

Description

Type library name	*JaguarTypeLibrary*
DLL name	*jagproxy.dll*

Defines symbolic constants for errors that can occur during server-side execution of a method call.

Table 4-3 lists the codes for server-side error numbers defined in the JagORBServerErrNum enumeration. User-defined IDL exceptions are not supported and are mapped to error number 9000.

Table 4-3: JagORBServerErrNum error codes

Symbolic Error Code	Number	Description
jagSrvMethExcepErr	9000	The method implementation threw a user-defined exception while executing in EAServer.
jagSrvMethInvalidErr	9001	The method name is either invalid or is presently not defined in the component's interface.
jagSrvMethInvalidArgErr	9002	The invocation of the component method failed because an invalid number of parameters was passed or a parameter type mismatch occurred.
jagSrvMethNotImplErr	9003	The invocation of the component method failed because the component does not implement the method.
jagSrvCompPermErr	9004	The invocation of the method in EAServer failed because user does not have the permissions to instantiate the component.
jagSrvCompDeployErr	9005	The invocation of the method in EAServer failed because component implementation was not deployed in EAServer.
jagSrvInternalErr	9006	The invocation of the method in EAServer failed due a fatal internal error.
jagSrvArgCountErr	9007	The invocation of the method in EAServer failed because an invalid parameter type was used by the method.
jagSrvSrvConnectErr	9008	The requested operation failed since the client cannot to acquire connection to the server.

Symbolic Error Code	Number	Description
jagSrvConversionErr	9009	The invocation of the method in EAServer failed due to a data conversion error.
jagSrvFreeMemErr	9010	The invocation of the method in EAServer failed while releasing memory resources.
jagSrvIntfReposErr	9011	The invocation of the method in EAServer failed while trying to access the interface repository.
jagSrvOutOfMemErr	9012	The invocation of the method in EAServer failed while trying to acquire memory from the operating system.
jagSrvOutOfResErr	9013	The invocation of the method in EAServer failed since it cannot acquire the necessary resources.
jagSrvSrvRespErr	9014	The invocation of the method in EAServer failed because there was no valid response from the server.
jagSrvInvObjrefErr	9015	The invocation of the method in EAServer failed because the object reference is invalid.

Usage

In Visual Basic, exceptions are mapped to the built-in Err object. The exception number maps to Err.Number and the description is available as Err.Description. You can handle exceptions by activating error handling code with On Error Goto statement or by checking whether Err.Number is > 0.

The proxy type library defines error numbers for client-side errors in the JagORBClientErrNum enumeration and server-side error numbers in the JagORBSrvErrNum enumeration.

See also

JagORBClientErrNum enumeration

JCollection interface

Description

Type library name	*JaguarTypeLibrary*
DLL name	*jagproxy.dll*

Represents a collection of objects or primitive data values; the ActiveX mapping for CORBA IDL sequences used as method parameter or return types in component methods.

Properties

Count as Integer
The number of available items.

Item(index as Integer) as Object
Set or retrieve an item's value. The first item has index 0. All VARIANT values are allowed, except for arrays. When setting item values, *index* can be any positive integer starting with 0, and the value can be any type supported by the ActiveX client proxy and ActiveX component dispatcher. See Chapter 20, "Creating ActiveX Clients," in the *EAServer Programmer's Guide* for a list of supported types.

Usage

JCollection represents an IDL sequence. Any return value or parameter that is defined as an IDL sequence in a EAServer IDL interface is represented as a JCollection in the equivalent ActiveX proxy interface. The JCollection contains the ActiveX equivalent for the base type of the IDL sequence. Nested IDL sequences map to nested JCollection instances.

Iterating over a collections items

You can iterate over the items in a JCollection instance using a For ... To loop or a For Each ... In loop. The following example shows a For ... To loop:

```
Dim stringJColl as JaguarTypeLibrary.JCollection

Set stringJColl = myComp.methodThatReturnsSequenceOfString()

Dim stringItem as String
Dim iter as Integer

For iter = 1 To stringJColl.Count
  stringItem = Format(stringJColl.Item(iter - 1))
Next iter
```

The following example shows a For Each ... In loop that iterates through all items in the collection *myJColl*:

```
Dim myJColl as JaguarTypeLibrary.JCollection

Set myJColl = myComp.methodThatReturnsSequenceOfString()

Dim myObject as Object
For Each myObject in myJColl
  Dim strItem as String
  strItem = format(MyObject)
```

```
Next
```

Nested collections

The following example shows how to iterate over items in a nested collection. In the example, *outerC* is a JCollection instance that contains JCollection instances as items:

```
Dim outerC as JaguarTypeLibrary.JCollection

set outerC = myComp.methodThatReturnsNestedSequence()

Dim innerC as JaguarTypeLibrary.JCollection
Dim strItem as String
Dim i as Integer
Dim j as Integer

For i=1 to outerC.Count
  innerC = outerC.Item(i - 1)
  For j=1 to innerC.Count
    strItem = Format(innerC.Item(j - 1))
  Next j
Next i
```

In a collection *innerC* nested inside another collection *outerC*, the *j*'th item in the *i*'th nested collection can be accessed directly as follows:

```
Dim outerC as JaguarTypeLibrary.JCollection

set outerC = myComp.methodThatReturnsNestedSequence()

Dim innerC as JaguarTypeLibrary.JCollection
Dim myObject as Object
i as Integer
Dim j as Integer

myObject = outerC.Item(i).Item(j)
```

See also Chapter 20, "Creating ActiveX Clients," in the *EAServer Programmer's Guide*

Object interface

Description

Type library name	*JaguarTypeLibrary*
DLL name	*jagproxy.dll*

A generic proxy object that must be narrowed to another interface.

Methods
- Narrow_ – Narrows the object to an instance of a named IDL interface.

See also SessionManager.Session, Orb.resolve_initial_references

Object.Narrow_

Description Narrows the object to an instance of a named IDL interface.

Syntax Object.Narrow_(idlName as String) as JaguarTypeLibrary.Object

Parameters
idlName
The name of the IDL interface to be narrowed to, in the form "*module*/*interface*", where *module* is the IDL module name and *interface* is the IDL interface name.

Return value An instance of the requested interface, which should be assigned to a variable declared as the equivalent ActiveX interface. An error is raised if the Object instance cannot support the requested interface.

Examples This example calls Orb.resolve_initial_references to obtain a proxy for the SSL service provider, then calls Object.Narrow_ to narrow to the CtsSecurity::SSLServiceProvider interface:

```
Dim orbRef As JaguarTypeLibrary.ORB
Dim ssp As CtsSecurity.SSLServiceProvider
Dim CORBAObj As Object

' Initialize the ORB
Set orbRef = New JaguarTypeLibrary.ORB
orbRef.Init ("")

' Get a proxy for the SSLServiceProvider
Set CORBAObj = _
    orbRef.resolve_initial_references("SSLServiceProvider")
Set ssp = CORBAObj.Narrow_("CtsSecurity/SSLServiceProvider")
```

Orb interface

Description

Type library name	*JaguarTypeLibrary*
DLL name	*jagproxy.dll*

The core interface used by clients that use CORBA-style proxy instantiation.

Methods

- Init – Initializes the Orb instance.
- resolve_initial_references – Obtains a proxy for a client-side service.
- object_to_string – Obtains a serialized string Interoperable Object Reference (IOR) for a proxy instance.
- string_to_object – Deserializes a string that contains a CORBA IOR representing a proxy for an EAServer component.

Usage

Orb is the core interface used by clients for CORBA-style proxy instantiation. In Visual Basic, construct an instance with the new keyword, as in this example:

```
Dim orbRef as JaguarTypeLibrary.Orb
set orbRef = new JaguarTypeLibrary.Orb
```

You can create multiple Orb instances, though there is no need to do so unless each is initialized differently by passing different properties to the Init method.

Orb.Init

Description

Initializes the Orb instance.

Syntax

Orb.Init (options as String)

Parameters

options

A string containing zero or more initialization parameter settings, formatted as follows:

```
orb.init("param1=setting1,param2=setting2")
```

As shown in the example, parameter names and values must be separated by an equals sign, '=', and each name/value pair must be separated from the next with a comma and no white space.

Usage

Init initializes an Orb instance. You must call Init once for each Orb instance before calling any other method. It is an error to call Init more than once on one instance. You can create several Orb instances and initialize them with different parameters.

Initialization Parameters

You can pass initialization parameters to the driver class by embedding settings in a formatted string, or setting environment variables. If both the environment variable and initialization parameter are set, the value of the initialization parameter is used. You can set any initialization parameter to a value of *none*, which overrides the value of the environment variable and sets the value to the default, if any.

You can pass settings for the following propertys to the driver class:

- **-ORBAXSSLCBComponent** Specifies the ProgID for an ActiveX component that implements the methods in the CtsSecurity.SSLCallback interface. When using SSL connections, you can install a callback to handle requests for required data, such as a certificate label or password, and exceptional conditions, such as server certificate signed by an unknown authority.
- **-ORBcertificateLabel** When using SSL, specifies the client certificate to use, if the server requests mutual authentication. The label is a simple name that identifies an X.509 certificate/private key in the Sybase PKCS #11 token or the Entrust token.
- **-ORBCodeSet** This sets the code set that the client uses. This parameter can also be set in an environment variable, JAG_CODESET. The default setting is *iso_1*.
- **-ORBentrustIniFile** When using SSL with an Entrust personal certificate, specifies the path name for the Entrust INI file that provides information on how to access Entrust. This is required when the useEntrustID property is set to true.
- **-ORBentrustUserProfile** When using SSL with an Entrust personal certificate, specifies an Entrust user profile path name. This property is optional when the Entrust single-login feature is available and required when this feature is not available.
- **-ORBentrustPassword** When using SSL with an Entrust personal certificate, specifies the password for logging in to Entrust with the specified user profile. This property is a null-terminated string, which is optional when the Entrust single-login feature is available and required when this feature is not available. If the password is required but not set, the getPin method in CtsSecurity.SSLCallback is invoked to get the Entrust password. If there is no callback or if the callback does not return a password, the SSL session fails.

- **-ORBHttp** Specifies whether the ORB should use HTTP-tunnelling to connect to the server. A setting of of "true" specifies HTTP tunnelling. The default is "false". This parameter can also be set in an environment variable, JAG_HTTP. Some firewalls may not allow IIOP packets through, but most all allow HTTP packets through. When connecting through such firewalls, set this property to “true”.
- **-ORBLogIIOP** Specifies whether the ORB should log IIOP protocol trace information. A setting of “true” enables logging. The default is "false". This parameter can also be set in an environment variable, JAG_LOGIIOP. When this parameter is enabled, you must set the ORBLogFile option (or the corresponding environment variable) to specify the file where protocol log information is written.
- **-ORBLogFile** Specifies the path and name of the file to which to log client execution status and error messages. This parameter can also be set in an environment variable, JAG_LOGFILE. There is no default; logging is not enabled unless you specify a filename to receive the log trace.
- **-ORBpin** When using SSL, specifies the PKCS #11 token PIN. This is required for logging in to a PKCS #11 token for client authentication and for retrieving trust information. If this property is not set and the server requests client authentication, the Login callback implementation is invoked to get the PKCS #11 PIN. If this property is set to the value `any`, then the getPin method in SSLCallback interface is invoked. If a PKCS #11 token login is required and neither the Login callback property nor the PIN property are set, the SSL session fails. This property can be set application-wide using the SSLServiceProvider context. This property cannot be retrieved once it has been set.
- **-ORBqop** When using SSL, specifies the name of a security profile characteristic. The security profile characteristic lists the CipherSuites the client uses when negotiating an SSL connection. If the qop is set, the ORB will connect only to listeners with an equal or greater level of security than required by the qop security profile. “Configuring security profiles” in the *EAServer System Administration Guide* describes the security characteristics that are provided with EAServer. At run time, you can retrieve a list of characteristics and their descriptions using the CtsSecurity.SSLServiceProvider interface. The default setting is “none”, which allows connections to listeners that do not use SSL at all.
- **-ORBProxyHost** Specifies the machine name or the IP address of an SSL proxy. There is no default.
- **-ORBProxyPort** Specifies the port number of the SSL proxy. There is no default.

- **-ORBRetryCount** Specifies the number of times to retry when the initial attempt to connect to the server fails. This parameter can also be set in an environment variable, JAG_RETRYCOUNT. The default is 5.
- **-ORBRetryDelay** Specifies the delay, in milliseconds, between retry attempts when the initial attempt to connect to the server fails.This parameter can also be set in an environment variable, JAG_RETRYDELAY. The default is 2000.
- **ORBSocketReuseLimit** Specifies the number of times that a network connection may be reused to call methods from one server. The default is 0, which indicates no limit. The default is ideal for short-lived clients. The default may not be appropriate for a long-running client program that calls many methods from servers in a cluster. If sockets are reused indefinitely, the client may build an affinity for servers that it has already connected to rather than randomly distributing its server-side processing load among all the servers in the cluster. In these cases, the property should be tuned to best balance client performance against cluster load distribution. In Sybase testing, a setting of 10 to 30 proved to be a good starting point. If the reuse limit is too low, client performance degrades.
- **-ORBuseEntrustID** When using SSL, specifies whether to use the Entrust ID or the Sybase PKCS #11 token for authentication. This is a Boolean (true or false) property. If this property is set to false, Sybase PKCS #11 token properties are valid and Entrust-specific properties are ignored. If this property is set to true, Entrust-specific properties are valid and Sybase PKCS #11 token properties are ignored. Entrust software is not included with EAServer, however, if your site uses Entrust for personal certificate management, this property allows you to connect to servers using Entrust certificates.
- **-ORBuserData** When using SSL, specifies user data (String datatype). This is an optional property. Client code can set user data during ORB initialization and access it using SSLSessionInfo.getProperty method in the SSL callback implementation. This may be useful as a mechanism to store ORB-level context information that is otherwise not available through the SSLSessionInfo interface.

Properties that configure SSL connections can also be set using the CtsSecurity.SSLServiceProvider interface, or by callback methods in a CtsSecurity.SSLCallback object that you install using the ORBAXSSLCBComponent property.

Example

This example creates an Orb instance and configures the `-ORBlogFile` property and the `-ORBpin` property, to specify a file name for logging errors and the Sybase SSL-certificate-database password, respectively:

```
Dim orb as JaguarTypeLibrary.Orb
set orb = new JaguarTypeLibrary.Orb
orb.init("-ORBlogFile=d:\jagorb.log,-ORBpin=sybase")
```

See also CtsSecurity.SSLServiceProvider interface

Orb.resolve_initial_references

Description Obtains a proxy for a client-side service.

Syntax
```
Orb.resolve_initial_references (
  serviceName as String
  ) as Object
```

Parameters *serviceName*

A string containing the name of the service. The following names are recognized:

Service name	Returned object
SSLServiceProvider	An instance of CtsSecurity.SSLServiceProvider.

Return value An Object (IDispatch pointer) that must be narrowed to the interface implemented by the service by calling the Object.Narrow_ method, as follows:

Service name	Type name for Object.Narrow_
SSLServiceProvider	CtsSecurity/SSLServiceProvider

Orb.object_to_string

Description Obtains a serialized string Interoperable Object Reference (IOR) for a proxy instance.

Syntax Orb.object_to_string(objRef as JaguarTypeLibrary.Object) as String

Parameters *objRef*

The proxy instance to be serialized. The instance must have been obtained from the EAServer ActiveX proxy server.

Return value A string that encodes the proxy object in CORBA IOR format.

Usage — object_to_string serializes a proxy object into a string, using the CORBA IOR format. You can call string_to_object to deserialize the object later.

See also — string_to_object

Orb.string_to_object

Description — Deserializes a string that contains a CORBA IOR representing a proxy for a EAServer comonent.

Syntax — Orb.string_to_object(ior as String) as Object

Parameters — *ior*

A string that was returned by object_to_string, or as a special case when obtaining a SessionManager.Manager instance, a URL formatted as follows:

```
protocol://host:port
```

Where protocol is `iiop` or `iiops` and `host:port` is the server's listener host address and port number. See SessionManager.Manager for more information.

Return value — An Object (IDispatch pointer) that must be narrowed to an instance of the appropriate interface by calling the Object.Narrow_ method.

Usage — string_to_object deserializes an object that was serialized using object_to_string.

The following restrictions apply when serializing and deserializing component proxy references:

- Unless the proxy is for an Enterprise Java EntityBean, the serialized reference remains valid only as long as the server has not been restarted since the time when proxy was first instantiated. When deserializing, the proxy instance will connect back to the same host and port as was used to create the original instance. An EntityBean proxy can be deserialized at any time, as long as the EntityBean is still installed on the original server.
- If the original proxy instance was created by connecting to a secure port with a client-side SSL certificate, the proxy must be deserialized in a session that connects using the same client certificate and equal or greater security constraints. For example, if you create an object with session that uses 128-bit SSL encryption, serialize the object, then later try to deserialize the object using during a session that uses 40-bit SSL encryption, the ORB will throw the CORBA::NO_PERMISSION exception. Access is allowed when objects created using less secure session are later accessed using a more secure session.

See also object_to_string

RecordSet interface

Description

Type library name	*JaguarTypeLibrary*
DLL name	*jagproxy.dll*

Represents a set of tabular data returned by a component method invocation. Provides methods to iterate through the rows in each result set. Modeled after RecordSet in Microsoft's ActiveX Data Objects (ADO) interface.

Properties

- Fields – Returns a Fields collection that contains a Field object for each column in the current row.
- EOF – Boolean. When tested after calling the MoveNext method, indicates whether the application has iterated over all rows in a result set. When tested after calling the NextRecordSet method, indicates whether the application has iterated through all available result sets.
- RecordCount – Integer. Specifies the number of rows in the current result set.

Methods

- MoveFirst – Positions the row pointer before the first row in the current result set.
- MoveNext – Moves the row pointer one row forward. Sets the EOF property to `true` if the row pointer has moved past the last row.
- NextRecordSet – Returns a RecordSet that represents the next result set that was returned by the method invocation. If all result sets have been viewed, returns an empty RecordSet and sets the EOF property to `true`.

Usage

RecordSet allows ActiveX client applications to retrieve result sets returned by a component method invocation. Each proxy component interface contains a GetRecordSet method. You can call this method after each method invocation to obtain a RecordSet object that contains the result sets returned by the method. If the method returned no result sets, GetRecordSet returns an empty RecordSet object. (You can test for this condition with the EOF property.)

For example code that uses RecordSet objects, see Chapter 20, "Creating ActiveX Clients," in the *EAServer Programmer's Guide*.

See also Fields collection, Field interface

RecordSet.MoveFirst

Description	Positions the row pointer before the first row in the current result set.
Syntax	RecordSet.MoveFirst()
Usage	Newly created RecordSet objects always have the row pointer positioned before the first record.
See also	MoveNext

RecordSet.MoveNext

Description	Moves the row pointer one row forward. Sets the EOF property to `true` if the row pointer has moved past the last row.
Syntax	RecordSet.MoveNext()
Usage	MoveNext is typically called in a loop while the EOF property tests as `true`.
See also	MoveFirst

RecordSet.NextRecordSet

Description	Returns a RecordSet that represents the next result set that was returned by the method invocation. If all result sets have been viewed, returns an empty RecordSet and sets the EOF property to `true`.
Syntax	RecordSet.NextRecordSet()
Usage	NextRecordSet is typically called in a loop until the EOF property tests as `true`.

CHAPTER 5

C Routines Reference

This chapter contains reference pages for the C routines that are provided for use by EAServer C or C++ components. Routines are indexed in the following sections:

Detailed reference pages for each routine follow the index sections. Routines are listed in alphabetical order by routine name.

Alphabetical list of all routines

- JagAlloc – Allocate memory for use in C component code.
- JagBeginResults – Begin the sequence of calls that sends a result set to the client.
- JagBindCol – Bind a memory address to a column in a result set.
- JagCmCacheProps – Retrieve connection cache properties.
- JagCmGetCachebyName – Retrieve the handle for the cache with the specified name.

- JagCmGetCachebyUser – Retrieve a cache handle for connections that use a specified set of values for server, user name, password, and connectivity library.
- JagCmGetConnection – Retrieve a connection from a specified cache or from any cache that matches a specified set of values for server, user name, password, and connectivity library.
- JagCmGetCtx – Obtain the connectivity-library-specific context reference that is used to allocate cached connections in a cache.
- JagCmGetProxyConnection – Retrieve a cached connection, specifying an alternate login name to set-proxy to.
- JagCmReleaseConnection – Place a connection back in the cache for reuse.
- JagColAttributes – Specify additional metadata for a column to be sent in a result set.
- JagCompleteWork – Indicate that the component's work for the current transaction was successfully finished and that this component instance should be deactivated.
- JagContinueWork – State indicator routine to specify that the component's work for the current transaction may be committed.
- JagDescribeCol – Describe a column to be sent as part of a result set.
- JagDisallowCommit – State indicator routine to specify that the current transaction cannot be committed because the component's work has not been completed.
- JagEndResults – Indicate that all rows in a result set have been sent.
- JagFree – Free memory that was allocated with JagAlloc.
- JagFreeCollectionHandle – Release the reference to a collection.
- JagFreeCollectionList – Release the memory allocated for the JagNameList structure.
- JagFreeSharedDataHandle – Release the shared variable handle.
- JagGetCollection – Retrieve a shared-data collection handle.
- JagGetCollectionList – Retrieve a list of all the collections defined in the server.
- JagGetHostName – Retrieve the client host name for the client connection that is associated with a C or C++ component instance.

- JagGetInstanceData – Retrieve the address of C component instance data.
- JagGetPassword – Retrieve the password for the client connection that is associated with a C or C++ instance.
- JagGetPeerAddress – Retrieve the client host IP address for the client connection that is associated with a C or C++ component instance.
- JagGetSharedData – Use the shared variable name to retrieve a shared variable handle.
- JagGetSharedDataByIndex – Use the shared variable index number to retrieve a shared variable handle.
- JagGetSharedValue – Retrieve a shared variable value.
- JagGetUserName – Retrieve the user name for the client connection that is associated with a C or C++ component instance.
- JagInTransaction – Determine whether the current method is executing in a transaction.
- JagIsRollbackOnly – Query whether the current transaction is doomed to be rolled back or is still viable.
- JagLockCollection – Lock a collection.
- JagLockNoWaitCollection – Lock a collection but do not wait for a locked collection to be unlocked.
- JagLog – Write a message to the server's log file.
- JagNewCollection – Create a shared-data collection or return a reference to an existing collection.
- JagNewSharedData – Create a shared variable with a specified name or retrieve the handle for the existing variable with the specified name.
- JagNewSharedDataByIndex – Create a shared variable with the specified index number or retrieve the existing variable with the specified index.
- JagResultsPassthrough – Forward results from an ODBC or Client–Library remote database command to the client.
- JagRollbackWork – Indicate that the component cannot complete its work for the current transaction. The component instance will be deactivated when the method returns.
- JagSendData – Send one row in a result set.

- JagSendMsg – Send an error message to the calling client application from a C or C++ component.
- JagSetInstanceData – Associate a reference to instance data with the current C component instance.
- JagSetSharedValue – Set a shared variable value.
- JagSleep – Suspend execution of the thread in which your component is running.
- JagUnlockCollection – Unlock a collection.

Routines for managing component instance data

These routines manage instance data in C components.

- JagGetInstanceData – Retrieve the address of C component instance data.
- JagSetInstanceData – Associate a reference to instance data with the current C component instance.

Routines for managing transaction flow

A component that participates in transactions can call these routines to influence the outcome of the current transaction. See Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide* for more information.

- JagCompleteWork – Indicate that the component's work for the current transaction was successfully finished and that this component instance should be deactivated when the method returns.
- JagContinueWork – Indicate that the component should not be deactivated after the current method invocation; allow the current transaction to be committed if the component instance is deactivated.
- JagDisallowCommit – Indicate that the current transaction cannot be committed because the component's work has not been completed; the instance remains active after the current method returns.
- JagInTransaction – Determine whether the current method is executing in a transaction.

- JagIsRollbackOnly – Query whether the current transaction is doomed to be rolled back or is still viable.
- JagRollbackWork – Indicate that the component cannot complete its work for the current transaction. The component instance will be deactivated when the method returns.

Routines for sharing data between components

These routines allow C or C++ components to share data.

- JagFreeCollectionHandle – Release the reference to a collection.
- JagFreeCollectionList – Release the memory allocated for the JagNameList structure.
- JagFreeSharedDataHandle – Release the shared variable handle.
- JagGetCollection – Retrieve a shared-data collection handle.
- JagGetCollectionList – Retrieve a list of all the collections defined in the server.
- JagGetSharedData – Use the shared variable name to retrieve a shared variable handle.
- JagGetSharedDataByIndex – Use the shared variable index number to retrieve a shared variable handle.
- JagGetSharedValue – Retrieve a shared variable value.
- JagLockCollection – Lock a collection.
- JagLockNoWaitCollection – Lock a collection but do not wait for a locked collection to be unlocked.
- JagNewCollection – Create a shared-data collection or return a reference to an existing collection.
- JagNewSharedData – Create a shared variable with a specified name or retrieve the handle for the existing variable with the specified name.
- JagNewSharedDataByIndex – Create a shared variable with the specified index number or retrieve the existing variable with the specified index.
- JagSetSharedValue – Set a shared variable value.
- JagUnlockCollection – Unlock a collection.

Routines for managing cached connections

For an overview of connection management, see “Chapter 26, “Using Connection Management,” in the *EAServer Programmer’s Guide*.

EAServer provides the following routines to manage cached connections:

- JagCmCacheProps – Retrieve connection cache properties.
- JagCmGetCachebyName – Retrieve the handle for the cache with the specified name.
- JagCmGetCachebyUser – Retrieve a cache handle for connections that use a specified set of values for server, user name, password, and connectivity library.
- JagCmGetConnection – Retrieve a connection from a specified cache or from any cache that matches a specified set of values for server, user name, password, and connectivity library.
- JagCmGetCtx – Obtain the connectivity-library-specific context reference that is used to allocate cached connections in a cache.
- JagCmGetProxyConnection – Retrieve a cached connection, specifying an alternate login name to set-proxy to.
- JagCmReleaseConnection – Place a connection back in the cache for reuse.

Routines for sending result sets

For information on how these routines are used to send results, see Chapter 25, “Sending Result Sets,” in the *EAServer Programmer’s Guide*.

- JagBeginResults – Begin the sequence of calls that sends a result set to the client.
- JagBindCol – Bind a memory address to a column in a result set.
- JagColAttributes – Specify additional metadata for a column to be sent in a result set.
- JagDescribeCol – Describe a column to be sent as part of a result set.
- JagEndResults – Indicate that all rows in a result set have been sent.
- JagResultsPassthrough – Forward results from an ODBC or Client-Library remote database command to the client.

- JagSendData – Send one row in a result set.

Routines for handling errors in C or C++ components

These routines are useful for handling errors in C components.

- JagSendMsg – Send an error message to the calling client application from a C component.
- JagLog – Write a message to the server's log file.

Routines for managing memory in C or C++ components

- JagAlloc – Allocate memory for use in C component code.
- JagFree - Free memory that was allocated with JagAlloc.

Routines to obtain user login information

You can call these routines in C or C++ component code to obtain information about the client connection that is associated with the current instance:

- JagGetHostName – Retrieve the client host name for the client connection that is associated with a C or C++ component instance.
- JagGetPassword – Retrieve the password for the client connection that is associated with a C or C++ component instance.
- JagGetPeerAddress – Retrieve the client host IP address for the client connection that is associated with a C or C++ component instance.
- JagGetUserName – Retrieve the user name for the client connection that is associated with a C or C++ component instance.

JagAlloc

Description Allocate memory for use in C component code.

Syntax

```
void * JAG_PUBLIC JagAlloc(
    SQLINTEGER len
    );
```

Parameters

len
The number of bytes to be allocated.

Return value

A pointer to newly allocated memory or NULL if the requested block of memory can not be allocated.

Usage

In C components, memory used to store output parameters for variable-length types (string and binary) must be allocated with JagAlloc.

Memory allocated with JagAlloc must be freed with JagFree.

In C++ components, use the standard CORBA memory allocation and deallocation routines.

See also

JagFree

JagBeginResults

Description

Begin the sequence of calls that sends a result set to the client.

Syntax

```
JagStatus JagBeginResults (
    SQLSMALLINT numColumns)
```

Parameters

numColumns
The number of columns in each row of the result set.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure. Check the server log file for error descriptions.

Usage

JagBeginResults is the first call in the sequence of calls that sends a result set to the client.

See also

Chapter 25, “Sending Result Sets,” in the *EAServer Programmer’s Guide*

JagBindCol

Description

Bind a program variable to a column in a result set.

Syntax

```
JagStatus JagBindCol(
        SQLSMALLINT  columnNumber,
        JagDataType  dataType,
        SQLSMALLINT  sourceType,
        SQLPOINTER   sourceBuf,
        SQLINTEGER   maxBuflen,
        SQLINTEGER   *buflen,
       SQLSMALLINT  *indicator)
```

Parameters

columnNumber

The column number to bind to. The first column is 1.

dataType

One of the following symbolic constants:

JAG_CS_TYPE	To indicate that Sybase Open Client Client-Library datatypes are being used.
JAG_ODBC_TYPE	To indicate that ODBC datatypes are being used.

sourceType

The *sql.h* type value that represents the C datatype of the bound variable. See the "Comments" section below for more information on datatypes.

sourceBuf

The memory address from which column values are to be read. Subsequent calls to JagSendData read values from the buffer. The *sourceBuf* address must remain valid until JagEndResults is called.

maxBuflen

The length, in bytes, of the *sourceBuf* buffer. For fixed-length types, *maxBuflen* is ignored.

buflen

The address of a SQLINTEGER variable that contains the length, in bytes, of the current value at the *sourceBuf* address. For columns with a variable-length datatype, JagSendData reads the length of the current value from **buflen*. The *buflen* address must remain valid until JagEndResults is called.

For fixed-length types, *buflen* is ignored.

indicator

The address of a SQLSMALLINT variable that acts as a null-indicator for column values. JagSendData reads this variable to determine whether the column value is null. The *indicator* address must remain valid until JagEndResults is called. Acceptable indicator values are:

Value	To indicate
CS_GOODATA or any value greater than or equal to 0.	Not null.
CS_NULLDATA	Null value.
SQL_NULL_DATA	Null value.

SQL_NULL_DATA and CS_NULLDATA can be used interchangeably.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagBindCol fails.

Usage

JagBindCol binds a program variable to a column in a result set. Binding associates the program variable with the result column: when JagSendData is called to send a row of data, it reads the current contents of the bound variable as the value of the column.

JagBindCol can use either ODBC or Open Client Client-Library datatypes. Set the *dataType* parameter to specify which set of type constants should be used.

ODBC datatypes

When the *dataType* parameter is JAG_ODBC_TYPE, JagBindCol interprets the *columnDatatype* parameter as an ODBC (*sql.h*) type constant. The C declaration of the bound variable must be an ODBC type that agrees with the C datatype. If necessary, JagSendData will perform conversion to the SQL datatype that was specified by JagDescribeCol. "C-to-SQL datatype conversions" on page 137 describes supported conversions between SQL datatypes and C datatypes.

Table 5-1 lists the ODBC C datatypes:

Table 5-1: ODBC C datatypes for JagBindCol

ODBC C type constant	ODBC type definition	Equivalent C declaration
SQL_C_CHAR	UCHAR *	unsigned char *
SQL_C_SSHORT	SWORD	short int
SQL_C_LONG	SDWORD	long int
SQL_C_SLONG	SDWORD	long int
SQL_C_ULONG	UDWORD	unsigned long int
SQL_C_FLOAT	SFLOAT	float
SQL_C_DOUBLE	SDOUBLE	double
SQL_C_BIT	UCHAR	unsigned char
SQL_C_STINYINT	SCHAR	signed char
SQL_C_UTINYINT	UCHAR	unsigned char
SQL_C_BINARY	UCHAR *	unsigned char *
SQL_C_DATE	DATE_STRUCT	struct { SQLSMALLINT year; SQLUINTEGER month; SQLUINTEGER day; } DATE_STRUCT;
SQL_C_TIME	TIME_STRUCT	struct { SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; } TIME_STRUCT;

SQL_C_TIMESTAMP	TIMESTAMP_STRUCT	struct {
		SWORD year;
		UWORD month;
		UWORD day
		UWORD hour;
		UWORD minute;
		UWORD second;
		UDWORD fraction;
		}
		fraction represents billionths of a second (1/1000000000)

C-to-SQL datatype conversions If the C datatype indicated by source type does not map directly to the column's SQL datatype (specified when JagDescribeCol was called), JagSendData will attempt to convert the value before sending it. The figure below shows which conversions are supported.

An X indicates a supported conversion.

SQL Datatype / C Datatype	SQL_C_CHAR	SQL_C_BIT	SQL_C_TINYINT	SQL_C_STINYINT	SQL_C_UTINYINT	SQL_C_SSHORT	SQL_C_SHORT	SQL_C_USHORT	SQL_C_SLONG	SQL_C_LONG	SQL_C_ULONG	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_BINARY	SQL_C_DATE	SQL_C_TIME	SQL_C_TIMESTAMP
SQL_CHAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SQL_VARCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SQL_LONGVARCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
SQL_DECIMAL	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_NUMERIC	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_BIT	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_TINYINT	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_SMALLINT	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_INTEGER	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_REAL	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_FLOAT	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_DOUBLE	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_BINARY	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_VARBINARY	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_LONGVARBINARY	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_DATE	X													X	X		X
SQL_TIME	X													X		X	X
SQL_TIMESTAMP	X													X	X	X	X

Client-Library datatypes

When the *dataType* parameter is JAG_CS_TYPE, JagBindCol interprets the *sourceDatatype* parameter as an Open Client Client-Library/C type constant. See your Client-Library documentation for descriptions of the Open Client datatypes. JagBindCol accepts any type constant that can be used with ct_bind except for CS_TEXT_TYPE and CS_IMAGE_TYPE. These types can be mapped to CS_LONGCHAR_TYPE and CS_LONGBINARY_TYPE, respectively.

See also

JagBeginResults, JagDescribeCol, JagSendData

Chapter 25, “Sending Result Sets,” in the *EAServer Programmer’s Guide*

JagCmCacheProps

Description

Retrieve connection cache properties.

Syntax

```
JagStatus JagCmCacheProps (
    JagCmCache          cache,
    JagAction           cmd,
    JagCmCachePropEnum  prop,
    SQLPOINTER          bufp,
    SQLINTEGER          buflen,
    SQLINTEGER          *outlen
    );
```

Parameters

cache

A JagCmCache control handle. You can call JagCmGetCachebyUser to obtain a cache handle for any cache that is defined in EAServer Manager. A non-null, valid cache handle is required to access any property other than JAG_CM_CACHEBYNAME.

When retrieving the JAG_CM_CACHEBYNAME value, you can pass a null cache handle, as described in "Determining whether by-name access is allowed" on page 141.

cmd

Must be JAG_GET.

prop

A symbolic constant that indicates the property of interest. Table 5-2 on page 140 lists possible values.

bufp

The address of a buffer or variable to receive the property value. Table 5-2 on page 140 lists the datatypes to pass for each property.

buflen

The length, in bytes, of **bufp*. If *buflen* indicates insufficient space for the value to be retrieved, JagCmCacheProps sets *outlen* to the required number of bytes and returns JAG_FAIL.

outlen

The address of a SQLINTEGER variable that receives the number of bytes written to **bufp*. For string properties, *outlen* includes the null-terminator.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

JagCmCacheProps fails for the following reasons:

- The *cache*, *cmd*, or *prop* parameters are invalid.
- *bufp* is NULL.

- *buflen* indicates that **bufp* contains insufficient space.

Check the server's log file for more information when JagCmCacheProps fails.

Usage

Table 5-2 summarizes connection cache properties. Access to all properties except JAG_CM_CACHEBYNAME requires a valid cache handle:

Table 5-2: Connection cache properties

Property	Specifies	*bufp datatype
JAG_CM_CACHEBYNAME	Whether a cache can be retrieved by calling JagCmGetCachebyName. This property can be accessed before retrieving a cache handle, as described in "Determining whether by-name access is allowed" on page 141.	A SQLCHAR array For input, specifies the cache name of interest. On return, unchanged.
JAG_CM_CACHENAME	The name of the cache (as it appears in EAServer Manager).	A SQLCHAR array. Input value is ignored. Output value is the cache name.
JAG_CM_CACHESIZE	The configured size of the cache.	SQLINTEGER Input is ignored. Output is the cache size.
JAG_CM_CONLIB	The connectivity library: "ODBC", "CTLIB_110", "OCI_7", or "OCI_8".	A SQLCHAR array. Input is ignored. Output is the connectivity library name.
JAG_CM_MUTEX	(UNIX only.) A Server-Library mutex that is used to single-thread ODBC calls. For more information, see Chapter 26, "Using Connection Management," in the *EAServer Programmer's Guide*.	A Server-Library SRV_OBJID mutex key. Input is ignored. Output is the mutex key.

Property	Specifies	*bufp datatype
JAG_CM_PASSWORD	The password used by connections in the cache.	A SQLCHAR array. Input is ignored. Output is the password.
JAG_CM_SERVER	The name of the server to which the cache's connections connect.	A SQLCHAR array. Input is ignored. Output is the server name.
JAG_CM_USERNAME	The user name for connections in the cache.	A SQLCHAR array. Input is ignored. Output is the server name.

Determining whether by-name access is allowed

The JagCmGetCachebyName method allows you to retrieve a connection cache by specifying only the cache name, rather than specifying values for the cache user name, password, and server name. However, by-name access must be enabled for the cache in EAServer Manager to allow retrieval with JagCmGetCachebyName.

You can call JagCmCacheProps to determine whether by-name access is allowed for a specified cache, before attempting to retrieve the cache handle with JagCmGetCachebyName. Pass the address of the cache name as the *bufp* parameter and the address of a SQLINTEGER for the *outlen* parameter. The *outlen value will be non-zero if the cache can be accessed with JagCmGetCachebyName. The example below illustrates the call syntax:

```
JagStatus       status;
SQLINTEGER      outval;
SQLCHAR         myCacheName[] = "mycache";

status = JagCmCacheProps((JagCMCache)NULL, JAG_GET,
                         JAG_CM_CACHEBYNAME,
                         (SQLPOINTER)myCacheName,
                         strlen(myCacheName),
&outval);
if (status != JAG_SUCCEED)
  ... log the error ...
}
```

```
if (outval == JAG_TRUE) {
  ... by-name access is not allowed for the cache ...
}
```

After retrieving a valid cache handle, you can determine whether by-name access is allowed as shown in the example below:

```
SQLINTEGER      outval;
JagCmCache      myValidCache;
JagStatus       status;

status = JagCmCacheProps(myValidCache, JAG_GET,
                         JAG_CM_CACHEBYNAME,
                         (SQLPOINTER)NULL,
                         0, &outval);
if (status != JAG_SUCCEED) {
  ... log the error ...
}
if (outval == JAG_TRUE) {
  ... by-name access is not allowed for the cache ...
}
```

See also JagCmGetCachebyUser, JagCmGetCtx

JagCmGetCachebyName

Description Retrieve the handle for the cache with the specified name.

Syntax

```
JagStatus JagCmGetCachebyName (
    SQLCHAR     *cachename,
    JagCmCache  *cache
    );
```

Parameters

cachename
The cache name.

cache
The address of a JagCmCache handle. If a matching cache is available, its handle is returned as **cache*. If no matching cache exists, **cache* is set to NULL.

Return value

Return value	To indicate
JAG_SUCCEED	Success. **cache* is set to the address of the matching cache.
JAG_FAIL	Failure.

JagCmGetCachebyName fails for the following reasons:

- A NULL value was passed for *cachename*.
- No matching cache was found.
- A matching cache is installed, but the cache properties do not allow retrieval with JagCmGetCachebyName. The "Enable cache-by-name access" option must be enabled in the Connection Cache Properties dialog.

JagCmGetCachebyName records a message that describes the failure reason in the server log file.

Usage

JagCmGetCachebyName allows you to retrieve connections without specifying the user name, password, and other parameters that are required by the JagCmGetCachebyUser routine.

You can retrieve a cache handle with either JagCmGetCachebyUser or JagCmGetCachebyName. Calling JagCmGetCachebyName allows you to change the cache user name, password, or server in EAServer Manager without requiring corresponding changes to your component source code.

In order for components to retrieve a cache with JagCmGetCachebyName, the EAServer Administrator must select the "Enable cache-by-name access" option for the cache in EAServer Manager. JagCmGetCachebyName fails if the cache does not have this option enabled.

Connection caches can be created, viewed, and modified with EAServer Manager. See Chapter 26, "Using Connection Management," in the *EAServer System Administration Guide* for details.

See also

JagCmGetCachebyUser

JagCmGetCachebyUser

Description

Retrieve a cache handle for connections that use a specified set of values for server, user name, password, and connectivity library.

Syntax

```
JagStatus JagCmGetCachebyUser (
    SQLCHAR     *username,
    SQLCHAR     *password,
    SQLCHAR     *server,
    SQLCHAR     *con_lib,
    JagCmCache  *cache
    );
```

Parameters

username
The user name for connections in the desired cache.

password
The password used by connections in the desired cache.

server
For ODBC connections, the ODBC data source name (as you would use to call SQLConnect). For Client-Library connections, the server name (as you would use to call ct_connect).

con_lib
A string value indicating the connectivity library used by connections in the cache. Allowable values are:

con_lib value	To indicate
"CTLIB_110"	Sybase Open Client Client-Library
"ODBC"	An ODBC implementation library
"OCI_7"	Oracle Call Interface 7.x
"OCI_8"	Oracle Call Interface 8.x

cache
The address of a JagCmCache handle. If a matching cache is available, its handle is returned as **cache*. If no matching cache exists, **cache* is set to NULL.

Return value

Return value	To indicate
JAG_SUCCEED	Success. **cache* is set to the address of the matching cache.
JAG_FAIL	Failure.

JagCmGetCachebyUser fails for the following reasons:

- A NULL value was passed for *username*, *password*, *server*, or *con_lib*.
- An invalid value was passed for *con_lib*.
- No matching cache was found.

Usage

JagCmGetCachebyUser allows you to retrieve connections that match the desired characteristic values for:

- Server name
- User name
- Password

- Connectivity library

You can use this routine when you are not sure if a cache is configured for a particular set of characteristic values. If no such cache is available, JagCmGetCachebyUser sets the **cache* parameter to NULL. If one or more matching caches exist, JagCmGetCachebyUser sets **cache* to the handle for the first matching cache that it finds.

Connection caches can be created, viewed, and modified with EAServer Manager. See Chapter 26, "Using Connection Management," in the *EAServer System Administration Guide* for details.

See JagCmGetConnection for an example that calls JagCmGetCachebyUser.

See also JagCmGetCachebyName

JagCmGetConnection

Description Retrieve a connection from a specified cache or from any cache that matches a specified set of values for server, user name, password, and connectivity library.

Syntax

```
JagStatus JagCmGetConnection (
    JagCmCache    *cache,
    SQLCHAR       *username,
    SQLCHAR       *password,
    SQLCHAR       *server,
    SQLCHAR       *con_lib,
    SQLPOINTER    *connection,
    JagCmOpt      opt
    );
```

Parameters

cache

The address of a JagCmCache cache handle variable. The input value determines how the parameter is used:

- If **cache* is not NULL, it must specify a valid cache handle. JagCmGetConnection attempts to return a connection from the specified cache. You can call JagCmGetCachebyUser to obtain a cache handle for any cache that is defined in EAServer Manager.
- If **cache* is NULL, characteristic values for *username*, *password*, *server*, and *con_lib* must be supplied. If a matching cache is found, **cache* is set to handle for the cache.

username
When **cache* is NULL, the user name for connections in the desired cache. Ignored when **cache* is not NULL.

password
When **cache* is NULL, the password used by connections in the desired cache. Ignored when **cache* is not NULL.

server
When **cache* is NULL, the name of the server to which cached connections are made. Ignored when **cache* is not NULL.

con_lib
When **cache* is NULL, indicates a string value indicating the connectivity library used by connections in the cache. Ignored when **cache* is not NULL.

When **cache* is NULL, allowable values for *con_lib* are:

con_lib value	To indicate
"CTLIB_110"	Sybase Open Client Client-Library
"ODBC"	An ODBC implementation library
"OCI_7"	Oracle Call Interface 7.x
"OCI_8"	Oracle Call Interface 8.x

connection
The address of a variable that receives the connection handle. Declare a variable of the appropriate type, as follows:

- For ODBC connections, pass the address of an SQLHDBC variable
- For Client-Library connections, pass the address of a CS_CONNECTION * variable
- For Oracle 7.x connections, pass the address of an OCI `Lda_Def` variable
- For Oracle 8.x connections, pass the address of an OCI `OCISvcCtx` variable

On successful return, the connection will be open and in a state that allows commands to be sent to the remote server.

opt

A symbolic value that indicates the desired behavior if all connections in a cache are in use. Allowable values are:

Value of opt	JagCmGetConnection behavior when all connections are in use
JAG_CM_NOWAIT	Fails with an error if no connection can be returned.
JAG_CM_WAIT	Does not return until a connection becomes available.
JAG_CM_FORCE	Allocates and opens a new connection. The new connection is not cached and will be destroyed when JagCmReleaseConnection is called.

Return value

Return value	To indicate
ODBC status code	The result of a SQLAllocConnect or SQLConnect call, or SQL_SUCCESS in the case where a previously opened connection is returned.
Client-Library status code	The result of a ct_con_alloc or ct_connect call, or CS_SUCCEED in the case where a previously opened connection is returned.
OCI_SUCCESS (An OCI 7.x and 8.x status code)	Successful retrieval of an OCI 7.x or 8.x connection.
OCI_FAIL (An OCI 7.x and 8.x status code)	Failure to retrieve an OCI 7.x or 8.x connection. Check the server log for errors, and verify that the connection can be pinged in EAServer Manager.
JAG_FAIL	Failure. JagCmGetConnection returns JAG_FAIL when the call specifies an invalid *con_lib* value.

Usage

JagCmGetConnection returns a connection that was allocated and opened with the specified connectivity library and that has matching values for server, user name, and password.

JagCmGetConnection behaves differently depending on whether the **cache* parameter is NULL.

Calls that pass a NULL cache handle

If **cache* is NULL, CmGetConnection looks for a cache with settings that match the values of the *username*, *password*, *server*, and *con_lib* parameters. If a cache is found and a connection is available, a connection is returned from that cache and **cache* is set to reflect the cache from which the connection came. If no cache is found, then a connection structure is allocated, a connection is opened using the specified connectivity library and the new connection structure is returned. If a cache was found, *con_lib* is ignored. The following table summarizes the JagCmGetConnection call when **cache* is NULL.

Table 5-3: JagCmGetConnection behavior when *cache is NULL

Cache found?	Connection available in cache?	Result
Yes	Yes	The call returns a connection handle in **connection* and sets **cache* to reflect the cache from which the connection came.
Yes	No	Depending on the value of the *opt* parameter, the call fails, waits for an available connection, or allocates and opens a new, uncached connection. **cache* is returned as NULL.
No	N/A	The call attempts to allocate and open a new, uncached connection. **cache* is returned as NULL.

Cached and uncached connections

A connection obtained with JagCmGetConnection is either cached or uncached.

A *cached connection* is one that was taken from a configured connection cache. When JagCmGetConnection returns a cached connection, it sets **cache* to indicate the cache to which the connection belongs. Cached connections must be released to the cache from which they were taken: pass the cache reference obtained in the JagCmGetConnection call when calling JagCmReleaseConnection.

An *uncached connection* is one that was not taken from a cache. JagCmGetConnection returns an uncached connection in either of the following cases:

- There is no cache configured with the specified *username*/*password*/*server*/*con_lib* parameter values.

- There is a matching cache, all its connections are in use, and the JagCmGetConnection call specifies JAG_CM_FORCE as the value of the *opt* parameter.

Calls that pass a non-NULL cache handle

When a cache handle is passed in **cache*, JagCmGetConnection looks for an available connection in that cache. If none is available, then the value of the *opt* parameter determines whether the call waits for a connection to be released, fails, or opens a new, uncached connection.

See also JagCmReleaseConnection

JagCmGetCtx

Description Obtain the connectivity-library-specific context reference that is used to allocate cached connections in a cache.

Syntax

```
JagStatus JagCmGetCtx (
    JagCmCache    *cache,
    SQLCHAR       *username,
    SQLCHAR       *password,
    SQLCHAR       *server,
    SQLCHAR       *con_lib,
    SQLPOINTER    *ctx
    );
```

Parameters

cache

The address of a JagCmCache cache handle variable. The input value determines how the parameter is used:

- When **cache* is NULL, the values of *username*, *password*, *server*, and *con_lib* are used to search for a matching cache. If found, **ctx* is set to the address of the connectivity-library context handle, and **cache* is set to the matching cache handle.
- If **cache* contains a valid cache handle, JagCmGetCtx retrieves the connectivity-library context for the indicated cache. You can call JagCmGetCachebyUser or JagCmGetCachebyName to obtain a cache handle for any cache that is defined in EAServer Manager.

username
When **cache* is NULL, the user name for connections in the desired cache. Ignored when **cache* is not NULL.

password
When **cache* is NULL, the password used by connections in the desired cache. Ignored when **cache* is not NULL.

server
When **cache* is NULL, the name of the server to which cached connections are made. Ignored when **cache* is not NULL.

con_lib
When **cache* is NULL, a string value indicating the connectivity library used by connections in the cache. Ignored when *cache* is not NULL.

When *cache* is NULL, *con_lib* must be one of the following:

con_lib value	To indicate
"CTLIB_110"	Sybase Open Client Client-Library
"ODBC"	An ODBC implementation library

ctx
The address of a variable that receives the connectivity library context used to allocate cached connections. The returned type depends on the connectivity library, as follows:

Connectivity library	Value returned in *ctx
Client-Library	A pointer to a CS_CONTEXT structure. Each connection cache uses a separate CS_CONTEXT structure.
ODBC	An ODBC SQLHENV environment handle. This handle is shared by all ODBC connection caches.

Return value

Returns	To indicate
JAG_SUCCEED	Successful retrieval of the CS_CONTEXT for a Client-Library connection cache.
JAG_FAIL	Failure. JagCmGetCtx fails when *con_lib* specifies an invalid value.

JagCmGetCtx fails for the following reasons:

- The *cache* parameter is passed as NULL.
- The value of *cache* is not NULL, and **cache* references an invalid cache.

- The value of *cache* is NULL, and there is no cache matching the values specified for the *username*, *password*, *server*, and *con_lib* parameters.

Usage

JagCmGetCtx retrieves the context or environment handle that is used to allocate connections in a cache.

See also

JagCmGetConnection

JagCmGetProxyConnection

Description

Retrieve a cached connection, specifying an alternate login name to set-proxy to.

Not all connection caches support set-proxy

JagCmGetProxyConnection cannot be used with OCI connections. You must be connected to a database server, such as Adaptive Server Enterprise 11.5, that supports the set session authorization command. Set-proxy support must be enabled for caches in EAServer Manager before you can use this feature. See Chapter 4, "Database Access," in the *EAServer System Administration Guide* for more information.

Syntax

```
JagStatus JAG_PUBLIC JagCmGetProxyConnection (
        JagCmCache      *cache,
        SQLCHAR         *username,
        SQLCHAR         *password,
        SQLCHAR         *server,
        SQLCHAR         *con_lib,
        SQLPOINTER      *connection,
        JagCmOpt        opt,
        SQLCHAR         *proxy
        );
```

Parameters

cache

The address of a JagCmCache cache handle variable. The input value determines how the parameter is used:

- When **cache* is NULL, the values of *username*, *password*, *server*, and *con_lib* are used to search for a matching cache. If found, **ctx* is set to the address of the connectivity-library context handle, and **cache* is set to the matching cache handle.
- If **cache* contains a valid cache handle, JagCmGetProxyConnection retrieves the connectivity-library context for the indicated cache. You can call JagCmGetCachebyUser or JagCmGetCachebyName to obtain a cache handle for any cache that is defined in EAServer Manager.

username

When **cache* is NULL, the user name for connections in the desired cache. Ignored when **cache* is not NULL.

password

When **cache* is NULL, the password used by connections in the desired cache. Ignored when **cache* is not NULL.

server

When **cache* is NULL, the name of the server to which cached connections are made. Ignored when **cache* is not NULL.

con_lib

When **cache* is NULL, a string value indicating the connectivity library used by connections in the cache. Ignored when *cache* is not NULL.

When *cache* is NULL, *con_lib* must be one of the following:

con_lib value	To indicate
“CTLIB_110”	Sybase Open Client Client-Library
“ODBC”	An ODBC implementation library

connection

The address of a variable that receives the connection handle. Declare a variable of the appropriate type, as follows:

- For ODBC connections, pass the address of an SQLHDBC variable
- For Client-Library connections, pass the address of a CS_CONNECTION * variable

On successful return, the connection will be open and in a state that allows commands to be sent to the remote server.

opt

A symbolic value that indicates the desired behavior if all connections in a cache are in use. Allowable values are:

Value of opt	JagCmGetConnection behavior when all connections are in use
JAG_CM_NOWAIT	Fails with an error if no connection can be returned.
JAG_CM_WAIT	Does not return until a connection becomes available.
JAG_CM_FORCE	Allocates and opens a new connection. The new connection is not cached and will be destroyed when JagCmReleaseConnection is called.

proxy

The user name to set-proxy to.

Return value

Return value	To indicate
ODBC status code	The result of a SQLAllocConnect or SQLConnect call, or the set session authorization command.
Client-Library status code	The result of a ct_con_alloc or ct_connect call, or the set session authorization command.
JAG_FAIL	Failure. JagCmGetConnection returns JAG_FAIL when the call specifies an invalid *con_lib* value.

Usage

JagCmGetProxyConnection retrieves a cached connection, specifying an alternate login name to set-proxy to. Set-proxy support must be enabled for a cache in EAServer Manager. If support is enabled, connections retrieved from the cache with JagCmGetConnection set-proxy to the client user name. Call JagCmGetProxyConnection to specify a different user name to set-proxy to.

Other than the set-proxy behavior, JagCmGetProxyConnection is identical to JagCmGetConnection.

See Chapter 4, “Database Access,” in the *EAServer System Administration Guide* for information on defining caches and enabling set-proxy support.

See also

JagCmGetConnection

JagCmReleaseConnection

Description

Place a connection back in the cache for reuse.

Syntax

JagStatus JagCmReleaseConnection (

```
JagCmCache    *cache,
SQLCHAR       *username,
SQLCHAR       *password,
SQLCHAR       *server,
SQLCHAR       *con_lib,
SQLPOINTER    connection,
SQLINTEGER    opt
);
```

Parameters

cache

The address of a JagCmCache cache handle variable. **cache* can be NULL or a valid cache handle.

If **cache* is not NULL, must be the cache handle that was used to obtain the connection by calling JagCmGetConnection.

If **cache* is NULL, JagCmReleaseConnection attempts to place the connection in a cache that has available space and that uses the same values for *username*, *password*, *server*, and *con_lib*. If no such cache has available space, the connection is closed and deallocated.

username

The user name of the connection. Ignored unless *cache* is NULL.

password

The password used by the connection. Ignored unless *cache* is NULL.

server

The name of the server to which the connection is made. Ignored unless *cache* is NULL.

con_lib

A string value indicating the connectivity library used by the connection. Ignored unless *cache* is NULL. Allowable values for *con_lib* are:

con_lib value	To indicate
“CTLIB_110”	Sybase Open Client Client-Library
“ODBC”	An ODBC driver library
“OCI_7”	Oracle Call Interface 7.x
“OCI_8”	Oracle Call Interface 8.x

connection

The connection handle to be released. The connection must be in a state that allows commands to be sent to the remote server. If commands were sent using the connection, the results of the commands must have been completely processed.

opt

One of the following symbolic constants:

opt value	To indicate
JAG_CM_DROP	The connection should be forced closed and deallocated. If the connection came from a cache, a new connection will be created in its place.
JAG_CM_UNUSED	Normal behavior: a connection taken from a cache is placed back in the cache; a connection created outside of a cache is closed and destroyed.

Use JAG_CM_DROP to destroy a connection when errors have made it unusable.

Return value

Returns	To indicate
ODBC or Client-Library return status	The result of connectivity library calls to close and deallocate a connection that was not released to a cache.
CS_SUCCEED	A Client-Library connection was returned to a cache.
SQL_SUCCESS	An ODBC connection was returned to a cache.
JAG_FAIL	Failure. JagCmReleaseConnection fails when *cache* is NULL and *con_lib* specifies an invalid value.

Usage

JagCmReleaseConnection releases control of a connection that was obtained from JagCmGetConnection.

Warning! Do not release a connection more than once.

See also

JagCmGetConnection

Chapter 26, “Using Connection Management,” in the *EAServer Programmer’s Guide* for examples.

JagColAttributes

Description

Specify additional metadata for a column to be sent in a result set.

Syntax

```
JagStatus JagColAttributes(
        SQLSMALLINT  item,
        SQLSMALLINT  descType,
        SQLPOINTER   descBuf,
        SQLINTEGER   buflen)
```

Parameters

item
The number of the column of interest. Column numbers start at 1.

descType
Must be SQL_COLUMN_MONEY.

descBuf
The address of a JagBoolean variable. Set the variable to JAG_TRUE to specify that the column represents a cash value; set the variable to JAG_FALSE otherwise. By default, columns do not represent a cash value.

buflen
The number of bytes in the *descBuf* buffer.

Usage

JagColAttributes specifies additional column attributes beyond those specified when JagDescribeCol is called. The only supported attribute is SQL_COLUMN_MONEY, which indicates that a column represents a cash value.

If you set the SQL_COLUMN_MONEY attribute to JAG_TRUE for a column, the column's values must be convertible to numeric values. Integer, floating-point, and fixed-point numeric data can be converted. Strings can be converted if the string values have the proper syntax to represent decimal numbers. Other datatypes can not be converted.

Note
If you are using Open Client datatypes with JagDescribeCol and JagBindCol, do not call JagColAttributes. Use the CS_MONEY datatype if a column represents a cash value.

See also

JagBindCol, JagDescribeCol

JagCompleteWork

Description

Indicate that the component's work for the current transaction has been successfully completed and is ready to be committed.

Syntax

```
void JagCompleteWork();
```

Usage

JagCompleteWork specifies that the component has successfully completed its contribution to the current transaction. The component instance deactivates when control returns from the current component method invocation.

If the component instance is the initiator of the transaction (that is, it was instantiated directly by a base client), then the component dispatcher attempts to commit the transaction. The transaction commits unless the commit is disallowed or vetoed; depending on the components that are participating, this can happen in any of the following ways:

- A participating C or C++ component has called JagDisallowCommit.
- A participating Java component throws an exception from its ServerBean.deactivate() method.
- A participating ActiveX component has called IObjectContext::disableCommit().

If a component is not transactional, then JagCompletework and JagRollbackWork have the same effect: both cause the component instance to deactivate after the currently executing method returns.

If a method calls none of JagCompleteWork, JagContinueWork, JagDisallowCommit, or JagRollbackWork, the default behavior is that of JagContinueWork.

See also

JagContinueWork, JagDisallowCommit, JagRollbackWork

Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide*

JagContinueWork

Description

Indicate that the component should not be deactivated after the current method invocation; allow the current transaction to be committed if the component instance is deactivated.

Syntax

```
void JagContinueWork();
```

Usage

JagContinueWork specifies that the component instance should not be automatically deactivated after the current method completes. If the instance is deactivated before the next method invocation, the current transaction is committed.

When a method calls JagContinueWork, the component instance is not deactivated until one of the following happens:

- The component's stub is destroyed explicitly by the client.
- The client disconnects without explicitly destroying the stub (the current transaction is always rolled back in this case).
- The component instance calls JagCompleteWork or JagRollbackWork during a subsequent method invocation.

JagContinueWork and JagDisallowCommit allow components that maintain state between method calls (using JagGetInstanceData and JagSetInstanceData). If a component is not transactional, JagContinueWork and JagDisallowCommit have the same effect: both prevent immediate deactivation of the component.

If a method calls none of JagCompleteWork, JagContinueWork, JagDisallowCommit, or JagRollbackWork, the default behavior is that of JagContinueWork.

See also

JagCompleteWork, JagDisallowCommit, JagRollbackWork

Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide*

JagDescribeCol

Description

Describe a column to be sent as part of a result set.

Syntax

```
JagStatus JagDescribeCol(
        SQLSMALLINT  item,
        JagDataType  dataType,
        SQLPOINTER   columnName,
        SQLSMALLINT  SQLDatatype,
        SQLUINTEGER  columnSize,
        SQLUINTEGER  precision,
        SQLSMALLINT  scale,
        SQLSMALLINT  nullable)
```

Parameters

item
The column number. Column numbers begin with 1.

dataType
One of the following symbolic constants:

JAG_CS_TYPE	To indicate that Sybase Open Client Client-Library datatypes are being used.
JAG_ODBC_TYPE	To indicate that ODBC datatypes are being used.

columnName
A null-terminated string containing the column's name.

columnDatatype
The ODBC or Client-Library type constant that indicates the column's datatype. See the "Comments" section below for more information on datatypes.

colLen
The maximum length for column values.

precision
The precision of column values. For SQL_NUMERIC or SQL_DECIMAL columns, *precision* indicates the maximum number of decimal digits that a value may have. For other datatypes, *precision* is ignored.

scale
The scale for column values. For SQL_NUMERIC or SQL_DECIMAL columns, *scale* indicates the number of decimal digits to the right of the decimal point. For other datatypes, *scale* is ignored.

nullable
One of the following symbolic constants:

Value	To indicate
SQL_NULLABLE	Column can contain null values.
SQL_NO_NULLS	Column values cannot be null.
SQL_NULLABLE_UNKNOWN	Equivalent to SQL_NULLABLE.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagDescribeCol fails.

Usage

JagDescribeCol describes the datatype and format of a column to be sent as part of a result set. The JagColAttributes routine specifies additional column metadata.

JagDescribeCol accepts either ODBC or Sybase Open Client type constants. Set the *dataType* parameter to specify which set of type constants should be used.

ODBC datatypes

When the *dataType* parameter is JAG_ODBC_TYPE, JagDescribeCol interprets the *columnDatatype* parameter as an ODBC (*sql.h*) type constant. The table below lists the supported ODBC SQL type constants. The first column is the SQL type constant and the second is the C datatype constant representing that type.

Table 5-4 describes the supported ODBC C datatypes.

Table 5-4: ODBC datatypes for JagDescribeCol

ODBC SQL type constant	Description
SQL_BINARY, SQL_VARBINARY, SQL_LONGBINARY	An array of bytes.
SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR	A string of one or more characters.
SQL_DECIMAL	A fixed point, fixed precision, fractional number.
SQL_NUMERIC	Same as SQL_DECIMAL.
SQL_SMALLINT	A 2-byte integer.
SQL_INTEGER	A 4-byte integer.
SQL_REAL	A 4-byte floating point value.
SQL_FLOAT	An 8-byte floating point value.
SQL_TIMESTAMP	An ODBC timestamp value. Timestamps are sent over the network in the same format as SQL_DATE.
SQL_DATE	A date value.
SQL_TIME	A time value.
SQL_BIT	A binary value.
SQL_TINYINT	A one-byte integer.

Client-Library datatypes

When the *dataType* parameter is JAG_CS_TYPE, JagDescribeCol interprets the *columnDatatype* parameter as an Open Client Client-Library/C type constant. JagDescribeCol accepts any type constant that can be used with ct_bind. See your Client-Library documentation for descriptions of these datatypes.

See also

JagBeginResults, JagBindCol, JagColAttributes, JagSendData

Chapter 25, "Sending Result Sets," in the *EAServer Programmer's Guide*

JagDisallowCommit

Description

Indicate that the current transaction cannot be committed because the component's work has not been completed; the instance remains active after the current method returns.

Syntax

```
void JagDisallowCommit();
```

Usage

JagDisallowCommit specifies that the component instance should not be automatically deactivated after the current method completes. If the instance is deactivated before the next method invocation, the current transaction is rolled back.

When a method calls JagDisallowCommit, the component instance is not deactivated until one of the following happens:

- The component's stub is destroyed explicitly by the client.
- The client disconnects without explicitly destroying the stub (the current transaction is always rolled back in this case).
- The component instance calls JagCompleteWork or JagRollbackWork during a subsequent method invocation.

JagContinueWork and JagDisallowCommit allow components to maintain state between method calls (using JagGetInstanceData and JagSetInstanceData). If a component is not transactional, JagContinueWork and JagDisableCommit have the same effect: both prevent immediate deactivation of the component.

If a method calls none of JagCompleteWork, JagContinueWork, JagDisallowCommit, or JagRollbackWork, the default behavior is that of JagContinueWork.

See also

JagCompleteWork, JagContinueWork, JagIsRollbackOnly, JagRollbackWork

Chapter 2, “Understanding Transactions and Component Lifecycles,” in the *EAServer Programmer’s Guide*

JagEndResults

Description

Indicate that all rows in a result set have been sent.

Syntax

```
JagStatus JagEndResults(SQLINTEGER rowCount)
```

Parameters

rowCount

The number of rows that were sent in the result set.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server’s log file for more information when JagEndResults fails.

Usage

JagEndResults indicates that all rows in a result set have been sent.

You must call JagEndResults after sending a result set.

See also

JagBindCol, JagDescribeCol, JagSendData

Chapter 25, “Sending Result Sets,” in the *EAServer Programmer’s Guide*.

JagFree

Description

Free memory that was allocated with JagAlloc.

Syntax

```
void JAG_PUBLIC JagFree(
    void *ptr
    );
```

Parameters

ptr

A pointer to the memory to be freed.

See also

JagAlloc

JagFreeCollectionHandle

Description
Release the reference to a collection.

Syntax
```
JagStatus JagFreeCollectionHandle (
        JagDataCollection * pCollection)
```

Parameters
pCollection
The address of the collection handle.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagFreeCollectionHandle fails.

Usage
This routine does not free any other resources besides the collection handle. See "Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also
JagFreeSharedDataHandle, JagGetCollection, JagNewCollection

JagFreeCollectionList

Description
Release the memory allocated for the JagNameList structure.

Syntax
```
JagStatus JagFreeCollectionList (
        JagNameList ** pList)
```

Parameters
pList
The pointer to the address of the JagNameList structure.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagFreeCollectionList fails.

Usage
See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also JagGetCollectionList

JagFreeSharedDataHandle

Description Release the shared variable handle.

Syntax
```
JagStatus JagFreeSharedDataHandle (
        JagSharedData * pData)
```

Parameters *pData*
The address of the shared variable handle.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagFreeSharedDataHandle fails.

Usage You must release the shared variable handle, otherwise a memory leak will occur.

Before releasing the shared variable handle, you must release the handle of the collection to which the shared variable belongs.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also JagFreeCollectionHandle, JagGetSharedData, JagGetSharedDataByIndex, JagNewSharedData, JagNewSharedDataByIndex

JagGetCollection

Description Retrieve a shared data collection handle.

Syntax
```
JagStatus JagGetCollection (
        SQLPOINTER name,
        JagDataCollection ** ppCollection)
```

Parameters *name*
The name of the collection.

ppCollection
The address of a JagDataCollection handle. *ppCollection* is set to NULL if the specified collection does not exist.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagGetCollection fails.

Usage

JagGetCollection retrieves a shared data collection handle. The collection must have been previously created by JagNewCollection.

Collections can be shared only among components that are installed in the same EAServer package. A collection created by a component that is installed in one package can not be retrieved by a component that is installed in a different package.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also

JagFreeCollectionHandle, JagGetCollection, JagLockCollection, JagLockNoWaitCollection, JagNewCollection, JagUnlockCollection

JagGetCollectionList

Description

Retrieve a list of all the collections defined in the server.

Syntax

```
JagStatus JagGetCollectionList (
        JagNameList ** pList)
```

Parameters

pList
A pointer to the address of the JagNameList structure.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagGetCollectionList fails.

Usage

The JagNameList structure is:

```
typedef struct _jagnamelist
{
    SQLINT            num_names;
    SQLPOINTER            *names;
} JagNameList;
```

where:

num_names is the number of array elements.

**names* is an array of *num_names* elements; each element points to a null-terminated collection name.

You must use the JagFreeCollectionList method to free the memory allocated for the JagNameList structure.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also JagFreeCollectionList

JagGetHostName

Description Retrieve the client host name for the client connection that is associated with a C or C++ component instance.

Syntax
```
JagStatus JAG_PUBLIC JagGetHostName(
    SQLPOINTER hostName,
    SQLINTEGER hostNameLen,
    SQLINTEGER *returnLen)
```

Parameters

hostName

The address of a character array to receive the client host name or, if the client software did not supply a host name, a zero-length string.

Java clients and JagGetHostName
Java clients do not supply the client host name (there is no mechanism to retrieve the host name in Java).

hostNameLen

The length, in bytes, of the *hostName* array. The length must include space for a null-terminator.

returnLen
NULL or the address of a SQLINTEGER variable.

returnLen is an optional output parameter that receives the length, in bytes, of the *hostName* value. The host name is null-terminated and the length includes the null-terminator.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

JagGetHostName fails for the following reasons:

- *hostName* was NULL.
- The buffer length is insufficient.
- The routine was called in code that was not executing in the context of a component method call.

Check the server's log file for more information when JagGetHostName fails.

See also JagGetPeerAddress

JagGetInstanceData

Description Retrieve the address of C component instance data.

Syntax
```
#include <jagpublic.h>
JagStatus JagGetInstanceData(CS_VOID **datapp);
```

Parameters
datapp
The address of a pointer to be set to the address of instance data. If no instance data has been installed, the pointer is set to NULL.

Return value JagGetInstanceData returns JAG_SUCCEED unless a serious error occurs, in which case JAG_FAIL is returned.

Usage JagSetInstanceData and JagGetInstanceData allow you to associate data with a particular instance of a C component. For example, you might save a counter and use it to keep track of how many times a particular method has been called.

JagSetInstanceData saves a pointer to component instance data; JagGetInstanceData retrieves the address of the saved data. For an introduction to these routines, see Appendix C, "Creating C Components," in the *EAServer Programmer's Guide*.

Note
To associate instance data with a C++ component, use class member variables.

See also

JagSetInstanceData

Appendix C, "Creating C Components," in the *EAServer Programmer's Guide*

JagGetPassword

Description

Retrieve the password for the client connection that is associated with a C or C++ component instance.

Syntax

```
JagStatus JAG_PUBLIC JagGetPassword(
        SQLPOINTER password,
        SQLINTEGER passwordLen,
        SQLINTEGER *returnLen)
```

Parameters

password

The address of a character array to receive the client password. If the connection has a NULL password, JagGetPassword writes a null-terminator to the *password* buffer.

passwordLen

The length, in bytes, of the *password* array. The length must include space for a null-terminator.

returnLen

NULL or the address of a SQLINTEGER variable.

returnLen is an optional output parameter that receives the length, in bytes, of the *password* value. The host name is null-terminated and the length includes the null-terminator.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

JagGetPassword fails for the following reasons:

- *password* was NULL.
- The buffer length is insufficient.
- The routine was called in code that was not executing in the context of a component method call.

Check the server's log file for more information when JagGetPassword fails.

See also JagGetHostName, JagGetUserName

JagGetPeerAddress

Description Retrieve the client host IP address for the client connection that is associated with a C or C++ component instance.

Syntax

```
JagStatus JAG_PUBLIC JagGetPeerAddress(
        SQLPOINTER peerAddress,
        SQLINTEGER bufLen,
        SQLINTEGER *returnLen)
```

Parameters

peerAddress
The address of a character array to receive the client IP address. The output value is "0.0.0.0" if the client's IP address is unavailable.

bufLen
The length, in bytes, of the *peerAddress* array. The length must include space for a null-terminator.

returnLen
NULL or the address of a SQLINTEGER variable.

returnLen is an optional output parameter that receives the length, in bytes, of the *peerAddress* value. The host name is null-terminated and the length includes the null-terminator.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

JagGetPeerAddress fails for the following reasons:

- *peerAddress* was NULL.

- The buffer length is insufficient.
- The routine was called in code that was not executing in the context of a component method call.

Check the server's log file for more information when JagGetPeerAddress fails.

See also JagGetHostName

JagGetSharedData

Description Use the shared variable name to retrieve a shared variable handle.

Syntax
```
JagStatus JagGetSharedData (
        JagDataCollection * pCollection,
        SQLPOINTER name,
        JagSharedData ** ppProp)
```

Parameters

pCollection
The handle of the collection to which the shared variable belongs.

name
The name of the shared variable.

ppProp
The shared variable handle. JagGetSharedData sets **ppProp* to NULL if the shared variable does not exist.

Return value

Return value	To indicate
JAG_SUCCEED	Success, even if the property does not exist
JAG_FAIL	Failure

Check the server's log file for more information when JagGetSharedData fails.

Usage This routine can retrieve only the handle of a property that has been created using the JagNewSharedData routine.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also JagFreeSharedDataHandle, JagGetSharedDataByIndex, JagNewSharedData, JagNewSharedDataByIndex

JagGetSharedDataByIndex

Description

Use the shared variable index number to retrieve a shared variable handle.

Syntax

```
JagStatus JagGetSharedData (
        JagDataCollection * pCollection,
        SQLINTEGER index,
         JagSharedData ** ppData)
```

Parameters

pCollection

The handle of the collection to which the shared variable belongs.

index

The index of the shared variable.

ppProp

The shared variable handle. **ppProp* is set to NULL if the shared variable does not exist.

Return value

Return value	To indicate
JAG_SUCCEED	Success, even if the property does not exist
JAG_FAIL	Failure

Check the server's log file for more information when JagGetSharedDataByIndex fails.

Usage

This routine can retrieve only the handle of a property that has been created using the JagNewSharedDataByIndex routine.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also

JagFreeSharedDataHandle, JagGetSharedData, JagGetSharedDataByIndex, JagNewSharedData

JagGetSharedValue

Description

Retrieve a shared variable value.

Syntax

```
JagStatus JagGetSharedData (
        JagSharedData * pData,
        SQLPOINTER buf,
        SQLINTEGER buflen,
        SQLINTEGER * outlen)
```

Parameters

pData
The shared variable handle.

buf
The buffer to which the shared variable value is to be copied.

buflen
The length, in bytes, of the buffer addressed by *buf*.

outlen
The address of a SQLINTEGER variable. On output, contains the length of the copied value. If no value has been set for the property, the length will be zero.

Return value

Return value	To indicate
JAG_SUCCEED	Success, even if there was no value to copy
JAG_FAIL	Failure

JagGetSharedValue fails if the size of the value is too large for the buffer.

Check the server's log file for more information when JagGetSharedValue fails.

Usage

You must create the buffer before you retrieve the shared variable value. Make sure the buffer is large enough to hold any value that can be stored in the shared variable.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also

JagGetSharedData, JagGetSharedDataByIndex, JagLockCollection, JagLockNoWaitCollection, JagNewSharedData, JagNewSharedDataByIndex, JagSetSharedValue, JagUnlockCollection

JagGetUserName

Description

Retrieve the user name for the client connection that is associated with a C or C++ component instance.

Syntax

```
JagStatus JAG_PUBLIC JagGetUserName(
        SQLPOINTER userName,
        SQLINTEGER userNameLen,
        SQLINTEGER *returnLen)
```

Parameters

userName
The address of a character array to receive the user name. The user name can have 0 length if no user name was supplied. In this case, only a null-terminator will be written to **userName*. (In practice, a user name is required to connect to the server unless user authentication is disabled.)

userNameLen
The length, in bytes, of the *userName* array. The length must include space for a null-terminator.

returnLen
NULL or the address of a SQLINTEGER variable.

returnLen is an optional output parameter that receives the length in bytes of the *userName* value. The user name is null-terminated and the length includes the null-terminator.

Return value

Return value	To indicate
JAG_SUCCEED	Success.
JAG_FAIL	Failure.

JagGetUserName fails for the following reasons:

- *userName* was NULL.
- The buffer length is insufficient.
- The routine was called in code that was not executing in the context of a component method call.

Check the server's log file for more information when JagGetUserName fails.

See also JagGetHostName, JagGetPassword

JagInTransaction

Description Determine whether the current method is executing in a transaction.

Syntax
```
JagBoolean JagInTransaction();
```

Usage Methods can call JagInTransaction to determine whether they are executing within a transaction. Methods in components that are declared to be transactional always execute as part of a transaction.

See also JagIsRollbackOnly

Chapter 2, “Understanding Transactions and Component Lifecycles,” in the *EAServer Programmer’s Guide*

JagIsRollbackOnly

Description
Query whether the current transaction is doomed to be rolled back or is still viable.

Syntax
```
JagBoolean JagIsRollbackOnly()
```

Return value
JAG_TRUE if the current transaction is doomed, in other words, it can never be committed. If executing outside of any transaction, returns JAG_FALSE.

Usage
Transactional components that issue intercomponent method calls should call JagIsRollbackOnly afterward to determine whether the current transaction is still viable. If not, the method should clean up and call JagRollbackWork to deactivate the current instance.

Transactions are doomed when a participating component has called JagRollbackWork (or its equivalent if the component is a Java or ActiveX component). Work performed by participating components is rolled back when the root component of the transaction deactivates.

See also
JagInTransaction, JagRollbackWork

Chapter 2, “Understanding Transactions and Component Lifecycles,” in the *EAServer Programmer’s Guide*

JagLockCollection

Description
Lock a collection.

Syntax
```
JagStatus JagLockCollection (
        JagDataCollection * pCollection)
```

Parameters
pCollection
The handle of the collection.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

JagLockCollection fails if the collection's isolation mode is JAG_LOCKDATA.

Check the server's log file for more information when JagLockCollection fails.

Usage

Locking a collection is strictly advisory. Even though a collection is locked, the JagGetSharedValue and JagSetSharedValue methods can still read and update the shared variables in the collection. If the collection is locked, JagLockCollection waits until the lock is released. To ensure that multiple read and update operations on any shared variable in a collection is atomic, lock the collection before executing read or update operations on the shared variables in the collection.

The JagLockCollection method prevents other JagLockCollection and JagLockNoWaitCollection requests from locking the collection until the lock is released. If the lock is successful, JAG_SUCCEED is returned. If the collection has already been locked by the calling object, this method does not lock the collection and JAG_SUCCEED is returned.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also

JagGetSharedValue, JagLockNoWaitCollection, JagSetSharedValue, JagUnlockCollection

JagLockNoWaitCollection

Description

Lock a collection but do not wait for a locked collection to be unlocked.

Syntax

```
JagStatus JagLockCollection (
        JagDataCollection * pCollection,
        JagBoolean * pLocked)
```

Parameters

pCollection

The handle of the collection.

pLocked

The address of a JagBoolean variable that will be set to indicate the lock status, as follows:

Value	To indicate
JAG_TRUE	The collection was not locked or the collection was already locked by the same calling object
JAG_FALSE	The collection was locked by another object

Return value

Return value	To indicate
JAG_SUCCEED	Success, even if the collection was locked by another object
JAG_FAIL	Failure

JagLockNoWaitCollection fails for the following reason:

- The collection's isolation mode is JAG_LOCKDATA.

Check the server's log file for more information when JagLockNoWaitCollection fails.

Usage

Locking a collection is strictly advisory. Even though a collection is locked, the JagGetSharedValue and JagSetSharedValue methods can still read and update the shared variables in the collection. If the collection is locked, JagLockNoWaitCollection does not wait until the lock is released and execution immediately returns to the calling method. To ensure that multiple read and update operations on any shared variable in a collection is atomic, lock the collection before executing read or update operations on the shared variables in the collection.

The JagLockNoWaitCollection method prevents other JagLockCollection and JagLockNoWaitCollection requests from locking the collection until the lock is released. If the lock is successful, JAG_SUCCEED is returned. If the collection has already been locked by the same calling object, this method does not lock the collection and JAG_SUCCEED is returned.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also

JagGetSharedValue, JagLockCollection, JagSetSharedValue, JagUnlockCollection

JagLog

Description

Write a message to the server's log file.

Syntax

```
#include <jagpublic.h>

JagStatus JagLog(
        JagBoolean use_date,
        SQLPOINTER logmsg)
```

Parameters

use_date

Pass as JAG_TRUE to indicate that the message should be preceded by a timestamp in the log; pass as JAG_FALSE to log the message without a timestamp.

logmsg

A null-terminated string containing the message to be logged. The message must include a newline at the end.

Return value

Return value	To indicate
JAG_SUCCEED	Success.
JAG_FAIL	Failure. JagLog fails if the log file can not be opened or if *logmsg* is NULL. If the log file cannot be opened, log messages are written to the server process' standard error device.

Usage

JagLog writes a message to the server's log file.

By convention, errors that occur on the server are written to the log. C or C++ components should use JagLog to record error messages in the log rather than printing to the console.

You can call JagSendMsg to send error messages to the client. When a method invocation fails you should log any details that will help debug the cause of failure, then send a descriptive error to the client.

For information on configuring the log file used by the server, see Chapter 3, "Creating and Configuring Servers," in the *EAServer System Administration Guide*.

See also

JagSendMsg

Appendix C, "Creating C Components," in the *EAServer Programmer's Guide*.

JagNewCollection

Description

Create a shared-data collection or return a reference to an existing collection.

Syntax

```
JagStatus JagNewCollection (
        SQLPOINTER name,
        JagLockLevel * pLockLevel,
        JagBoolean * pExists,
        JagDataCollection ** ppCollection)
```

Parameters

name
The name of the collection.

pLockLevel

Value	To indicate
JAG_LOCKCOLLECTION	Allows locks to be set on collections
JAG_LOCKDATA	Does not allow locks to be set on collections

pExists
is set to one of the following values:

Return value	To indicate
JAG_TRUE	If a collection with the specified name already exists
JAG_FALSE	If a collection with the specified name is created

ppCollection
A pointer to the address of the collection handle.

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagNewCollection fails.

Usage

The JagNewCollection method:

- Creates a new collection with the specified name and lock level, returns a reference to that collection, and sets **pExists* to JAG_FALSE, or
- Returns a reference to the existing collection with the specified name and sets **pExists* to JAG_TRUE. The method's lock level is ignored and the collection's current lock level is returned in **pLockLevel*.

Collections can be shared only among components that are installed in the same EAServer package. A collection created by a component that is installed in one package can not be retrieved by a component that is installed in a different package.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also

JagFreeCollectionHandle, JagFreeSharedDataHandle, JagGetCollection, JagLockCollection, JagLockNoWaitCollection, JagUnlockCollection

JagNewSharedData

Description

Create a shared variable with a specified name or retrieve the handle for the existing variable with the specified name.

Syntax

```
JagStatus JagNewSharedData (
        JagDataCollection * pCollection,
        SQLPOINTER name,
        JagBoolean * pExists,
        JagSharedData ** ppProp)
```

Parameters

pCollection

The handle of the collection in which you want to create the shared variable.

name

The name of the shared variable.

ppProp

The shared variable handle.

pExists

The address of a JagBoolean status variable. **pExists* is set to one of the following values:

Return value	To indicate
JAG_TRUE	If a shared variable with the specified name already exists
JAG_FALSE	If a shared variable with the specified name is created

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagNewSharedData fails.

Usage

The JagNewSharedData creates a shared variable with the specified name or returns a reference to the existing shared variable. Newly created variables are initialized to NULL and a reference to the new variable is returned.

Shared variables are either named or indexed. Named variables are created with JagNewSharedData and retrieved with JagGetSharedData. Indexed variables are created with JagNewSharedDataByIndex and retrieved with JagGetSharedDataByIndex.

Named shared variables are uniquely identified by the collection which contains them (see JagGetCollection) and the name assigned when the property is created with JagNewSharedData.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also

JagFreeSharedDataHandle, JagGetSharedData, JagGetSharedDataByIndex, JagGetSharedValue, JagNewSharedDataByIndex, JagSetSharedValue

JagNewSharedDataByIndex

Description

Create a shared variable with the specified index number or retrieve the existing variable with the specified index.

Syntax

```
JagStatus JagNewSharedData (
        JagDataCollection * pCollection,
        SQLINTEGER index,
        JagBoolean * pExists,
        JagSharedData ** ppProp)
```

Parameters

pCollection

The handle of the collection in which you want to create the shared variable.

index

An integer that uniquely identifies the shared variable within the collection. *index* can be any number within the range of the SQLINTEGER datatype.

ppProp

The shared variable handle.

pExists

The address of a JagBoolean status variable. **pExists* is set to one of the following values:

Return value	To indicate
JAG_TRUE	If a shared variable with the specified index number already exists
JAG_FALSE	If a shared variable with the specified index number is created

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagNewSharedDataByIndex fails.

Usage

The JagNewSharedDataByIndex creates a shared variable with the specified index number or returns a reference to the existing shared variable with that index. Newly created variables are initialized to NULL and a reference to the new variable is returned.

Shared variables are either named or indexed. Indexed variables are created with JagNewSharedDataByIndex and retrieved with JagGetSharedDataByIndex. Named variables are created with JagNewSharedData and retrieved with JagGetSharedData.

Indexed shared variables are uniquely identified by the collection which contains them (see JagGetCollection) and the index assigned when the property is created with JagNewSharedDataByIndex.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also

JagFreeSharedDataHandle, JagGetSharedData, JagGetSharedDataByIndex, JagGetSharedValue, JagNewSharedData, JagSetSharedValue

JagResultsPassthrough

Description

Forward results from an ODBC or Client-Library remote database command to the client.

Syntax

```
#include <jagpublic.h>

JagResultsPassthrough(
        JAGPOINTER     conlib,
        JAGPOINTER     conlib_ptr,
        JagPthruType   pthruType)
```

Parameters

conlib

One of the following strings:

- "ODBC" to indicate that *conlib_ptr* is the address of an ODBC HSTMT control structure.
- "CTLIB" to indicate that *conlib_ptr* is the address of a Client-Library CS_COMMAND control structure.

conlib_pointer
The address of the control structure used to access result rows for the connectivity library that you are using.

When using Client-Library, set *conlib_ptr* to the address of a CS_COMMAND structure. The CS_COMMAND structure must be in a state that allows ct_results to be called without error.

When using ODBC, set *conlib_ptr* to the address of an HSTMT control structure. The HSTMT must be in a state that allows SQLFetch to be called without error.

pthruType
One of the following symbolic constants to indicate how results are to be processed:

pthruType value	To indicate
JAG_PTHRU_ALL_RESULTS	All results from the current command will be retrieved and sent to the client.
JAG_PTHRU_CURRENT_RESULTS	Only rows from the current result set will be returned.

Return value

Return value	To indicate
JAG_SUCCEED	Successfully sent results.
JAG_NO_MORE_RESULTS	Applies only when using Client-Library and the JAG_PTHRU_CURRENT_RESULTS option for *pthruType*. Indicates that all results have been retrieved from the CS_COMMAND structure.
JAG_FAIL	Failure

Check the server's log file for more information when JagSendMsg fails.

Usage

JagResultsPassthru forwards results from an ODBC or Client-Library remote database command to the client.

All results from a query can be forwarded with one call using the JAG_PTHRU_ALL_RESULTS option for the *pthruType* parameter. To forward single result sets, use the JAG_PTHRU_ALL_RESULTS option.

When using the JAG_PTHRU_ALL_RESULTS option with Client-Library, any result type other than row results (CS_ROW_RESULTS) causes JagResultsPassthrough to fail.

When forwarding single result sets, you must ensure that you retrieve or cancel all results. The sections below describe the loop algorithms for forwarding individual result sets.

Forwarding individual result sets with Client-Library

When using the JAG_PTHRU_CURRENT_RESULTS option with Client-Library, call JagResultsPassthrough in place of calling ct_results. JagResultsPassthrough returns JAG_NO_MORE_RESULTS when CS_COMMAND structure. The code fragment below illustrates how JagResultsPassthrough can be called in a loop:

```
JagStatus   jagRet;
CS_RETCODE  retcode;
CS_CHAR     *sqlCmd = "select * from titles select * f
rom authors"
CS_COMMAND      *cmd;

// Deleted the code which did CT-Lib
// initialization, connected to the SQL Server,
// and allocated the CS_COMMAND structure.

retcode = ct_command(cmd, CS_LANG_CMD, sqlCmd,
                     CS_NULLTERM, CS_UNUSED);
if (retcode != CS_SUCCEED)
{
    // handle failure
}
retcode = ct_send(cmd);
if (retcode != CS_SUCCEED)
{
    // handle failure
}
while ((jagRet = JagResultsPassthrough("CTLIB", cmd,
        JAG_PTHRU_CURRENT_RESULTS)) == JAG_SUCCEED)
{
    // No code needed here. JagResultsPassthru
    // did all the work
      ;
}
if (jagRet != JAG_NO_MORE_RESULTS)
{
    // handle failure
}
```

Forwarding individual result sets with ODBC

When using the JAG_PTHRU_CURRENT_RESULTS option with ODBC, call JagResultsPassthrough before calling SQLMoreResults, instead of the usual SQLFetch row processing. The code fragment below illustrates how JagResultsPassthrough and SQLMoreResults can be called in a loop to forward all result sets to the client.

```
RETCODE     odbcRet;
CS_CHAR     *sqlCmd =
        "select * from titles select * from authors"
HSTMT       hstmt;

// Deleted the code which did ODBC initialization,
// connected to the SQL Server, and allocated
// the HSTMT.

odbcRet = SQLExecDirect(hstmt, (SQLCHAR *)sqlCmd, SQL_
NTS);
if (odbcRet != SQL_SUCCESS)
{
    // handle failure
}
do
{
    jagRet = JagResultsPassthrough("ODBC", &hstmt,
                    JAG_PTHRU_CURRENT_RESULTS);
    if (jagRet != JAG_SUCCEED)
    {
        // handle failure
    }
} while (SQLMoreResults(hstmt) == SQL_SUCCESS);
if (odbcRet != SQL_NO_DATA_FOUND)
{
    // handle failure
}
```

See also

JagBindCol, JagDescribeCol, JagEndResults

Chapter 25, “Sending Result Sets,” in the *EAServer Programmer’s Guide*

JagRollbackWork

Description

Indicate that the component cannot complete its work for the current transaction. The component instance will be deactivated when the method returns.

Syntax

void JagRollbackWork();

Usage

JagRollbackWork specifies that the component cannot complete its work for the current transaction. The transaction will be rolled back when the initiating component is deactivated.

Calling JagRollbackWork does not automatically cause the current method invocation to fail. Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* describes how to code graceful method failures.

If a component is not transactional, then JagRollbackWork and JagRollbackWork have the same effect: both cause the component instance to deactivate after the currently executing method returns.

If a method calls none of JagCompleteWork, JagContinueWork, JagDisallowCommit, or JagRollbackWork, the default behavior is that of JagContinueWork.

See also

JagCompleteWork, JagContinueWork, JagDisallowCommit, JagInTransaction, JagIsRollbackOnly

Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide*

JagSendData

Description

Send one row in a result set.

Syntax

JagStatus JagSendData()

Return value

Return value	To indicate
JAG_SUCCEED	Success
JAG_FAIL	Failure

Check the server's log file for more information when JagSendData fails.

Usage

JagSendData sends a row of data to the client. Data for the columns in the row is copied from the program variables bound with JagBindCol.

After you have sent all rows with JagSendData, you must call JagEndResults to indicate the end of the result set.

See Chapter 25, “Sending Result Sets,” in the *EAServer Programmer’s Guide* for more information on sending result sets.

See also JagBindCol, JagDescribeCol, JagEndResults

JagSendMsg

Description Send an error message to the calling client application from a C component.

Syntax

```
#include <jagpublic.h>

JagStatus JagSendMsg(
        JagSeverity severity,
        SQLINTEGER errnum,
        SQLPOINTER msgtext)
```

Parameters

severity
Must be JAG_SEVERITY_ERROR.

errnum
An integer code for the error.

msgtext
A null-terminated string containing a description of the error.

Return value JAG_SUCCEED for successful execution.

If an error occurs, JagSendMsg writes error descriptions to the server log file and returns JAG_FAIL.

Usage JagSendMsg sends an error message to the client application that invoked the currently executing method. JagSendMsg provides a C facility similar to Java exceptions.

Note
Do not call JagSendMsg in C++ components. Instead, throw a user-defined or CORBA system exception.

JagSendMsg should be called only to describe errors that prevent successful completion of a method call. JagSendMsg causes an exception to be thrown in a Java or ActiveX client; in these clients, the exception may preempt the arrival of INOUT parameter values. To return additional status information from a successful call, use additional INOUT parameters.

JagSendMsg should be called only once per method execution, because clients may not be able to process more than one message.

How clients process messages

Clients process the received message differently depending on the type of client:

- JagSendMsg and Java Clients

  For Java clients, a JagSendMsg call on the server causes the stub method call to throw a Java exception on the client. Note that a component can call JagSendMsg multiple times, however, a Java client receives an exception for only the first call. The message is embedded in a client exception as follows:

  - If the active method's definition has a raises clause that lists an exception that contains a string, the client stub throws an instance of that exception. The exception's string field contains the message text.
  - Otherwise, the client stub throws an instance of org.omg.CORBA.UNKNOWN that contains the message text.

- JagSendMsg and ActiveX Clients

  For ActiveX clients, a JagSendMsg call on the server causes an ActiveX automation exception on the client.

See also

JagLog

Appendix C, "Creating C Components," in the *EAServer Programmer's Guide*

JagSetInstanceData

Description

Associate a reference to instance data with the current C component instance.

Syntax

```
#include <jagpublic.h>
JagStatus JagSetInstanceData(CS_VOID *datap);
```

Parameters

datap
A pointer to instance data.

Return value

JagSetInstanceData returns JAG_SUCCEED unless a serious error occurs, in which case JAG_FAIL is returned.

Usage

JagSetInstanceData and JagGetInstanceData allow you to associate data with a particular instance of a C component. For example, you might save a counter and use it to keep track of how many times a particular method has been called.

JagSetInstanceData saves a pointer to component instance data; JagGetInstanceData retrieves the address of the saved data. For an introduction to these routines, see Appendix C, "Creating C Components," in the *EAServer Programmer's Guide*.

Note
To associate instance data with a C++ component, use class member variables.

See also

JagGetInstanceData

Appendix C, "Creating C Components," in the *EAServer Programmer's Guide*

JagSetSharedValue

Description

Set a shared variable value.

Syntax

```
JagStatus JagSetSharedData (
        JagSharedData * pData,
        SQLPOINTER pValue,
        SQLINTEGER len)
```

Parameters

pData
The shared variable handle.

pValue
A buffer containing the new value.

len
The size (in bytes) of the value. If the value is a null-terminated string, you must include the length of the null terminator in the length of the string.

Return value

Return value	To indicate
JAG_SUCCEED	Success

Return value	To indicate
JAG_FAIL	Failure

Check the server's log file for more information when JagSetSharedData fails.

Usage

The JagSetSharedValue method copies a value to a specified shared variable. You must have retrieved the shared variable reference before executing this method. You must pass a pointer to the value you want to copy to the shared variable. You must specify the size of the value.

There are two possible strategies for using shared data values:

- Pass a pointer to the value so that JagSetSharedValue copies the new value.

  This approach is easier to implement, however it should not be used for sharing large data values. Repeated copying of large values can adversely affect performance.

- Pass the address of a pointer to the value, so that JagSetSharedValue copies only the address of memory that contains the value.

  Use this approach for data structures that contain pointers or for large values. When you use this approach, you must allocate and free the memory used to store shared values yourself. Memory must be allocated with malloc or its equivalent; do not use local variables.

  When using this approach, your component must always lock the property during the time that the property data is in use to ensure that the data is not overwritten or freed while it is in use. Locking can be achieved one of two ways:

  - Lock the collection – Create the collection with the *pLockLevel* option set to JAG_LOCKCOLLECTION when calling JagNewCollection. Call JagLockCollection or JagLockNoWaitCollection to lock the collection.
  - Use system calls for locking – Use system calls to create a semaphore or mutex that is stored with the data. You can use the semaphore or mutex to prevent concurrent access.

  The first approach is preferable because it is simpler to implement and portable to different platforms.

Note
JagSetSharedValue does not follow pointers in structures when copying data. If the shared variable is a structure that contains pointers, then only the addresses are copied, not the memory contents at those addresses.

Example storing data values directly

The code below calls JagSetSharedValue to save "tombstone" as a shared data value. The *len* parameter is passed as 1 byte more than the string length to ensure that the null-terminator is copied:

```
SQLCHAR buf[20];
JagSharedData  *pData

strcpy(buf, "tombstone");
retcode = JagSetSharedValue(pData, buf, strlen(buf) +
1);
```

Example storing pointers to shared data

The code below allocates a SQLCHAR pointer, then calls JagSetSharedValue to save the pointer as shared data.

```
SQLCHAR        *ptrToData;
JagSharedData  *pData

ptrToData = (SQLCHAR *)malloc(20);
strcpy(ptrToData, "tombstone");

/*
** Pass the address of the pointer to save the pointer;
** the length of the shared data is the size of the
** pointer
*/

retcode = JagSetSharedValue(pData,
                            &ptrToData,
sizeof(ptrToData));
```

Here is code to retrieve the value that was set in the example above:

```
SQLCHAR        *ptrToData;
JagSharedData  *pData
SQLINTEGER     outlen;

retcode = JagGetSharedValue(pData, &ptrToData,
                            sizeof(ptrToData),
&outlen);
```

See also

JagGetSharedData, JagGetSharedDataByIndex, JagGetSharedValue, JagNewSharedData, JagNewSharedDataByIndex

Appendix C, "Creating C Components," in the *EAServer Programmer's Guide*

JagSleep

Description

Suspend execution of the thread in which your component is running.

Syntax

```
void JAG_PUBLIC JagSleep (
        JagLong seconds)
```

Parameters

seconds
The number of seconds to sleep.

Usage

JagSleep suspends execution of the thread in which the current component instance is running. JagSleep is useful in service components that perform background processing in the run method. run typically loops forever, and calling JagSleep prevents your component from dominating the server's CPU execution time.

JagSleep can only be called by a component that is executing within EAServer. This routine is not available to clients.

Warning! In EAServer components, never call the sleep system routine or any other routine that suspends execution of the current process. Doing so suspends execution of the server. JagSleep suspends only the current thread, allowing components running in other threads to continue execution.

JagUnlockCollection

Description

Unlock a collection.

Syntax

```
JagStatus JagUnlockCollection (
        JagDataCollection * pCollection)
```

Parameters

pCollection
The handle of the collection to unlock.

Return value

Return value	To indicate
JAG_SUCCEED	Success, even if the calling method has not locked the collection
JAG_FAIL	Failure

Check the server's log file for more information when JagUnlockCollection fails.

Usage

The JagUnlockCollection method releases a lock on a collection. A locked collection is automatically released when the object's method execution is completed. However, to make your application more efficient and prevent deadlocks, unlock a collection when you are finished updating or reading the shared variable in the collection so that other objects can access the collection right away.

See Appendix C, "Creating C Components," in the *EAServer Programmer's Guide* for more information.

See also

JagLockCollection, JagLockNoWaitCollection

APPENDIX A

Deprecated Java Classes and Interfaces

This appendix documents obsolete EAServer Java classes and interfaces, which are based on an obsolete version (version 0.4) of the Enterprise Java Beans specification.

Rather than using these models for developing Java components, use the following:

- The latest EJB version, for portability to other J2EE based application servers.
- The CORBA component model, for compatibility with CORBA based application servers. If using CORBA, you can achieve lifecycle semantics similar to the EJB model by configuring the component to use the control interface CtsComponents::ObjectControl. For documentation of this interface, see the following file in your EAServer installation directory:

  ```
  html/ir/CtsComponents__ObjectControl.html
  ```

Package Index

com.sybase.jaguar.beans.enterprise

Classes and interfaces used to implement Java components and to create stubs for remote communication. These classes are based on an early draft of the Enterprise JavaBeans specification. Future releases of the Java Developer's Kit will likely provide built-in classes with the same functionality:

- jaguar.beans.enterprise.EnterpriseBeanException class – Exception that can be thrown by components that implement the ServerBean interface..
- jaguar.beans.enterprise.InstanceContext interface – An InstanceContext object allows a Java component to influence the outcome of the transaction in which it is participating.
- jaguar.beans.enterprise.ServerBean interface – Interface for EAServer Java components, with methods that support transactional behavior and reuse of component instances.
- jaguar.beans.enterprise.SharedObjectException class – Class representing exceptions that are thrown by SharedObjects interface methods.
- jaguar.beans.enterprise.SharedObjects interface – Interface to support sharing data between instances of the same component.

jaguar.beans.enterprise.EnterpriseBeanException class

Description	package com.sybase.jaguar.beans.enterprise; public class JCM extends Exception Exception that can be thrown by components that implement the ServerBean interface.
Constructors	Same as java.lang.Exception.
Methods	Same as java.lang.Exception.
See also	ServerBean

jaguar.beans.enterprise.InstanceContext interface

Description	package com.sybase.jaguar.beans.enterprise; public interface InstanceContext extends Object

An InstanceContext object allows a Java component to influence the outcome of the transaction in which it is participating. A component method's calls to the InstanceContext state primitives also determine the component's state after the method completes. See "ServerBean lifecycle" on page 200 for more information.

Constructors

None. A component that implements the ServerBean interface receives an InstanceContext object as a parameter to the method activate(InstanceContext, String). A component that does not implement the ServerBean interface can call Jaguar.getInstanceContext() to obtain an InstanceContext object.

Methods

- completeWork() – For transactional components, indicate that the transaction in which a component is participating should be committed. For any component, indicate that the instance should be deactivated.
- continueWork() – Indicate that the current component instance cannot be deactivated automatically when control returns from the current component method invocation.
- getSharedObjects() – Get a SharedObjects object that allows access to data shared among instances of a component.
- inTransaction() – Determine whether the current component instance is executing in the context of a transaction.
- isRollbackOnly() – Determine if the current transaction is doomed.
- rollbackWork() – For transactional components, indicate that the transaction in which a component is participating should be aborted and rolled back. For any component, indicate that the instance should be deactivated.

Usage

See Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide* for a description of how components participate in transactions.

See also

jaguar.beans.enterprise.ServerBean interface, jaguar.beans.enterprise.SharedObjects interface

InstanceContext.completeWork()

Description

For transactional components, indicate that the transaction in which a component is participating should be committed. For any component, indicate that the instance should be deactivated.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	InstanceContext

```
public abstract void completeWork();
```

Usage

For a transactional component, completeWork() indicates that the component's contribution to the current transaction has been successfully completed. For any component, completeWork() indicates that the component instance should be deactivated when control returns from the current component method invocation.

If the component is transactional and the component instance is the initiator of the transaction (that is, it was instantiated directly by a base client), then EAServer attempts to commit the transaction. The transaction commits unless the commit is vetoed. Depending on the components that are participating, a veto can happen in any of the following ways:

- A participating Java component throws an exception from its ServerBean.deactivate() method.
- A participating C component has called JagDisallowCommit.
- A participating ActiveX component has called IObjectContext.disableCommit().

If the component instance is not the initiator of the transaction, the transaction may be rolled back when another participating instance calls rollbackWork() in addition to any of the cases listed above.

You can call completeWork(), continueWork(), and rollbackWork() many times in one method. Only the last call to execute takes effect. If you call none of these, the default behavior is that specified by continueWork().

See also

continueWork(), rollbackWork(), isRollbackOnly(), inTransaction()

InstanceContext.continueWork()

Description

Indicate that the current component instance cannot be deactivated automatically when control returns from the current component method invocation.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	InstanceContext

```
public abstract void continueWork();
```

Usage

Calling continueWork() indicates that the component instance should not be deactivated when the method returns. The component instance is not deactivated until one of the following happens:

- The transaction times out or the client's instance reference expires. In either case, the current transaction is rolled back.
- The transaction's root component calls completeWork() or rollbackWork(). If your component implements the ServerBean interface, it can veto the transaction by throwing an exception in the deactivate() method.
- The component instance calls completeWork() or rollbackWork() during a subsequent method invocation.

You can call completeWork(), continueWork(), and rollbackWork() many times in one method. Only the last call to execute takes effect. If you call none of these, the default behavior is that specified by continueWork().

See also

completeWork(), rollbackWork(), isRollbackOnly(), inTransaction()

InstanceContext.getSharedObjects()

Description

Get a SharedObjects object that allows access to data shared among instances of a component.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	InstanceContext

```
public abstract SharedObjects getSharedObjects();
```

See also

jaguar.beans.enterprise.SharedObjects interface

InstanceContext.inTransaction()

Description

Determine whether the current component instance is executing in the context of a transaction.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	InstanceContext

```
public abstract boolean inTransaction();
```

Return value — `true` if the current component instance is executing as part of a transaction; `false` otherwise.

Usage — Java component methods can call inTransaction() to determine whether they are executing within a transaction. Methods in components that are declared to be transactional always execute as part of a transaction. See Chapter 2, "Understanding Transactions and Component Lifecycles," in the *EAServer Programmer's Guide* for more information.

See also — completeWork(), continueWork(), isRollbackOnly(), rollbackWork()

InstanceContext.isRollbackOnly()

Description — Determine if the current transaction is doomed.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	InstanceContext

```
public abstract boolean isRollbackOnly();
```

Return value — `true` if the current transaction is doomed; `false` if the transaction is in a committable state or if the current component instance is not executing as part of a transaction.

Usage — Call isRollbackOnly() to determine whether the current transaction is still viable.

If a component participates in a multi-component transaction, you should call isRollbackOnly() in the following places:

- After issuing intercomponent calls
- At the start of methods that can be executed by intercomponent calls.

If the transaction is no longer viable, there is no point in continuing execution. The method should clean up and call rollbackWork() to deactivate the component instance.

See also — completeWork(), continueWork(), inTransaction(), rollbackWork()

InstanceContext.rollbackWork()

Description — For transactional components, indicate that the transaction in which a component is participating should be aborted and rolled back. For any component, indicate that the instance should be deactivated.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	InstanceContext

```
public abstract void rollbackWork();
```

Usage

For a transactional component, rollbackWork() indicates that the component cannot complete its contribution to the current transaction. After the method returns, the transaction is doomed: the transaction flow continues until all participating components are deactivated. At that point, the transaction is rolled back.

In any component, rollbackWork() indicates that the component instance should be deactivated when control returns from the current component method invocation.

You can call rollbackWork(), continueWork(), and completeWork() many times in one method; only the last call to execute takes effect. If you call none of these, the default behavior is that specified by continueWork().

Transactional components that make intercomponent method calls can call isRollbackOnly() to determine whether the current transaction is still viable or has been set to rollback only.

See also

completeWork(), continueWork(), inTransaction(), isRollbackOnly()

jaguar.beans.enterprise.ServerBean interface

Description

```
package com.sybase.jaguar.beans.enterprise;
public interface ServerBean
```

Interface for EAServer Java components, with methods that support transactional behavior and reuse of component instances.

Constructors

None required. If a component's implementation class provides a default constructor, the EAServer runtime server calls the default constructor when creating a new component instance.

Methods

- activate(InstanceContext, String) – Indicates that this component instance has been activated.
- canReuse() – Specify whether this component instance is eligible for reuse.
- deactivate() – Indicates that this component instance has been deactivated.

- destroy() – Indicates that this component instance is being released and will not be activated again.

Usage

A component that implements ServerBean can participate in instance pooling. The server can maintain a cache of idle component instances and bind them to individual clients only as needed. This strategy allows the server to service more clients without the performance drain caused by allocating a component instance for each request.

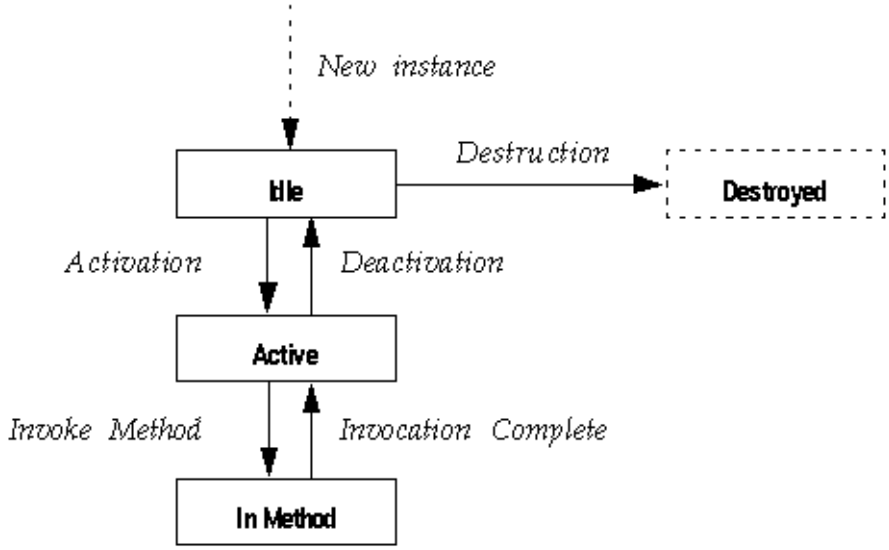
The activate(InstanceContext, String) method indicates that an instance is being removed from the pool to service a client. The deactivate() method indicates that the instance is finished servicing the client. Instance reuse is optional (see “Support for instance pooling” on page 202). However, components that support it will achieve greater scalability.

The ServerBean lifecycle is tightly coupled with the EAServer transaction model. See Chapter 2, “Understanding Transactions and Component Lifecycles,” in the *EAServer Programmer’s Guide* for a description of how components participate in transactions.

ServerBean lifecycle

Figure A-1 illustrates the states and state transitions in the lifecycle of a Java component that implements ServerBean.

Figure A-1: States in the ServerBean lifecycle



The state transitions are as follows:

- *New instance* – The EAServer runtime allocates a new instance of the component class. The default constructor is called if one exists. The instance remains idle until the first method invocation.
- *Activation* – Activation prepares a component instance for use by a client. activate(InstanceContext, String) is called. Once an instance is activated, it is bound to one client and can service no other client until it has been deactivated.
- *In Method* – In response to a method invocation request from the client, the EAServer runtime calls the corresponding class method in the component. The next state depends on the method's execution, as follows:
  - If the method throws an uncaught exception, the instance is deactivated. If the method is participating in a transaction, the transaction is rolled back.
  - If the method has called InstanceContext.rollbackWork() or InstanceContext.completeWork(), the instance is deactivated.

    - If the method has called InstanceContext.continueWork(), the instance is not deactivated. The client's next method invocation is serviced by the same instance unless the client destroys its reference or disconnects.

- *Deactivation* – Deactivation occurs when:

    - The instance has called either InstanceContext.rollbackWork() or InstanceContext.completeWork()
    - The current transaction times out, or
    - The client's instance reference has expired.

    The EAServer runtime calls the component's deactivate() method to indicate deactivation.

    You can define your component so that instances are recycled after deactivation, as described in "Support for instance pooling" on page 202.

- *Destruction* – The EAServer runtime calls destroy() to indicate that references to the class instance are being released. The instance is deallocated at a later time by the Java garbage collector thread.

Support for instance pooling

Instance pooling allows a single component instance to be activated and deactivated many times to serve different clients. Instance pooling can increase the performance of your application, since it eliminates unnecessary instance allocations. There are two ways to support pooling:

- In EAServer Manager, you can configure your component so instances are always pooled by selecting the Pooling option on the Instances tab in the Component Properties window.
- Alternatively, you can implement the ServerBean.canReuse() method to specify at runtime whether an instance can be pooled. If canReuse() returns `true`, the instance is pooled. Otherwise, the instance is destroyed.

If the component's Pooling option is enabled in EAServer Manager, EAServer never calls the canReuse() method since instances are always pooled.

If your component supports pooling, you must add code to the activate(InstanceContext, String) method that resets any class variables to their initial values. When activate returns, the component state must be the same as if the component were freshly constructed. If the component keeps references to stateful objects across activation cycles, you must reset these objects to an initial state as well.

See also InstanceContext

ServerBean.activate(InstanceContext, String)

Description

Indicate that this component instance has been activated.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	ServerBean

```
public abstract void activate
        (InstanceContext ctx, String instanceKey)
        throws EnterpriseBeanException;
```

Parameters

ctx

An InstanceContext that is associated with the current component instance. activate should save a reference to the instance context for use in later method calls. This reference becomes invalid and must be discarded when deactivate() is called.

instanceKey

Not used.

Usage

activate and deactivate allow a component's instances to be pooled. If a component supports instance pooling, activate must reset any class variables to the initial values, as if the component instance were being freshly constructed. To prohibit instance pooling, code the canReuse() method to return `false`.

See "ServerBean lifecycle" on page 200 for more information on when activate and deactivate are called.

If a component is declared to be transactional and its activate method throws an exception, the EAServer runtime server rolls back the transaction in which the component is about to participate.

See also

deactivate(), canReuse()

ServerBean.canReuse()

Description

Specify whether this component instance is eligible for reuse.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	ServerBean

```
public abstract boolean canReuse()
```

Return value

`true` or `false` to indicate whether the component instance is eligible to be recycled.

Usage

If the Pooling option is not set for your component in EAServer Manager, EAServer calls the component's canReuse method after deactivating each instance to determine whether the instance can be reused. If canReuse returns `false`, EAServer destroys the instance. If the Pooling option is set, EAServer never calls the canReuse method. For more information on component properties, see the EAServer Manager online help.

Components that support instance pooling must be coded such that a recycled instance behaves the same as a newly allocated instance. Your implementation of the activate(InstanceContext, String) method must ensure that the instance state is reset to that of a newly allocated instance.

See also

activate(InstanceContext, String), deactivate(), destroy()

ServerBean.deactivate()

Description

Indicates that this component instance has been deactivated.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	ServerBean

```
public abstract void deactivate()
        throws EnterpriseBeanException;
```

Usage

The EAServer runtime calls deactivate() to indicate that the component instance is being deactivated. See "ServerBean lifecycle" on page 200 for more information on when activate and deactivate are called.

If your component caches data changes, you can code the deactivate() method to send cached changes to the remote database server. deactivate() can call InstanceContext.isRollbackOnly() to determine whether the current transaction is being committed or rolled back. If the transaction is being committed, deactivate() must send any cached database changes to the remote server(s).

If deactivate() throws an exception, the current transaction (if any) is rolled back; the caller of the component method that attempted to commit the transaction receives the exception as a JException with the message text included.

If your component is transactional and it maintains state (it calls InstanceContext.continueWork() from one or more methods), then deactivate() must verify that the current component state is ready for commit and throw an exception if it is not.

Note
deactivate should release references to the InstanceContext object that was received in the activate(InstanceContext, String) method. The InstanceContext is meaningless after deactivate has been called.

See also activate(InstanceContext, String), canReuse(), destroy()

ServerBean.destroy()

Description Indicates that this component instance is being released and will not be activated again.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	ServerBean

```
public abstract void destroy();
```

Usage destroy should release any resources that were allocated by the component's constructor.

See also activate(InstanceContext, String), deactivate(), canReuse()

jaguar.beans.enterprise.SharedObjectException class

Description

```
package com.sybase.jaguar.beans.enterprise;

public class SharedObjectException
        extends Exception
```

Class representing exceptions that occur during SharedObjects processing.

Constructors Same as java.lang.Exception.

Methods Same as java.lang.Exception.

See also SharedObjects

jaguar.beans.enterprise.SharedObjects interface

Description

```
package com.sybase.jaguar.beans.enterprise;
public interface SharedObjects
```

Interface to support sharing data between instances of the same component.

Constructors

None. See InstanceContext.getSharedObjects(), ServerBean.activate(InstanceContext, String).

Methods

- get(int) – Retrieve the value of a property.
- lock(int) – Place an advisory lock on a property.
- lockNoWait(int) – Place an advisory lock on a property. If the property is currently locked, do not wait for the current lock to be released and execution immediately returns to the calling method.
- set(int, Object) – Set the value of a property.
- unlock(int) - Unlock a property locked by the same instance executing the unlock method.

See also

jaguar.beans.enterprise.InstanceContext interface

SharedObjects.get(int)

Description

Retrieve the value of a property.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	SharedObjects

```
public abstract Object get
        (int index)
        throws SharedObjectException;
```

Parameters

index

An arbitrary integer that identifies the property from which you want to retrieve the value.

Usage

To retrieve a property value, retrieve an object reference to the property using the get method and then assign the object reference to a variable with the desired datatype. If the property has not been initialized, the property and variable are initialized to null.

Executing a single get method on a property is atomic. *Atomic* means that an operation on data will complete before any other operations can access that data.

See also set(int, Object), lock(int), lockNoWait(int), unlock(int)

SharedObjects.lock(int)

Description Place an advisory lock on a property.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	SharedObjects

```
public abstract void lock
        (int index)
        throws SharedObjectException;
```

Parameters *index*

An integer that identifies the property you want to lock.

Usage Use the lock method in combination with the lockNoWait and unlock methods to synchronize multiple updates to and reads from the same property value. The lock method places an advisory lock on a property. An *advisory lock* prevents another instance from locking the property but does not prevent another instance from using the get and set methods to retrieve and update the property value. If the property is currently locked, the lock method waits for the current lock to be released.

You must lock a property before using the get or set method to retrieve or update the property value. When you lock a property that has not been set, the property is created and its value is initialized to `null`. You can lock the same property more than once as long as all locks are executed from the same component instance. However, these multiple locks are not iterative and you only have to unlock the property once.

See also lockNoWait(int), unlock(int), get(int), set(int, Object)

SharedObjects.lockNoWait(int)

Description Place an advisory lock on a property. If the property is currently locked, do not wait for the current lock to be released and execution immediately returns to the calling method.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	SharedObjects

```
public abstract void lockNoWait
        (int index)
        throws SharedObjectException;
```

Parameters

index

An integer that identifies the property you want to lock.

Usage

Use the lockNoWait method in combination with the lock and unlock methods to synchronize multiple updates to and reads from the same property value. The lockNoWait method places an advisory lock on a property. An *advisory lock* prevents another instance from locking the property but does not prevent another instance from using the get and set methods to retrieve and update the property value. If the property is currently locked, the lockNoWait method does not wait for the current lock to be released and execution immediately returns to the calling method.

You must lock a property before using the get or set method to retrieve or update the property value. When you lock a property that has not been set, the property is created and its value is initialized to `null`. You can lock the same property more than once as long as all locks are executed from the same component instance. However, these multiple locks are not iterative and you only have to unlock the property once.

See also

lock(int), unlock(int), get(int), set(int, Object)

SharedObjects.set(int, Object)

Description

Set the value of a property.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	SharedObjects

```
public abstract Object set
        (int index)
        Object obj)
       throws SharedObjectException;
```

Parameters

index

An integer that identifies the property for which you want to set a value.

obj
An object containing the new property value.

Usage

To set a property value, assign a value an object and pass that object as the *obj* parameter in the set method.

Executing a single set method on a property is atomic. That is, the call will complete before any other operations can access the property being set.

See also

get(int), lock(int), lockNoWait(int), unlock(int)

SharedObjects.unlock(int)

Description

Unlock a property locked by the same instance executing the unlock method.

Syntax

Package	com.sybase.jaguar.beans.enterprise
Interface	SharedObjects

```
public abstract void unlock
        (int index)
        throws SharedObjectException
```

Parameters

index
An integer that identifies the property to be locked.

Usage

Use the unlock method in combination with the lock and lockNoWait methods to synchronize multiple updates to and reads from the same property value. The unlock method releases an advisory lock on a property that has been locked by the instance executing the unlock method. An *advisory lock* prevents another instance from locking the property but does not prevent another instance from using the get and set methods to retrieve and update the property value.

You can unlock a property that has not been set. Even if a property has been locked more than once, you only have to unlock the property once.

See also

lock(int), lockNoWait(int), get(int), set(int, Object)

Index

B

C

D

F

G

I

J

L

N

O

R

S

T

W